FIDO: A SYSTEM THAT CAN BE TAUGHT TO RESPOND WITH SUITABLE OUTPUT MESSAGES

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Master of Science in Computer Science
in the
University of Canterbury
by
C. V. Collister

University of Canterbury
1979

ACKNOWLEDGEMENTS

While preparing this thesis I have discovered, as I suppose many others have, that a thesis is not the work of any one person but is the work of many. To list everyone who has helped me in presenting this thesis would be to list a great number of people. However without the help of some people this thesis could never been completed:

My father, brothers and friends, people who have little technical interest in the project but were a great source of encouragement. Professor Penny who encouraged me to do a Masters degree, conceived the basic idea for the project, discussed ideas and critically read drafts of my thesis. Mr. R. Harries with whom I spent many hours discussing ideas, who patiently read drafts of my thesis and acted as my supervisor while Professor Penny was away. Other staff of the Computer Science Department who acted as sounding boards for ideas and gave me support in so many ways. Other students with whom the interchange of ideas provided valued assistance.

ABSTRACT

An investigation has been made into improving messages output by a computer system by using a program called FIDO (From Input Derive Output), which can be taught to give appropriate output messages in varying situations. A number of possible applications for such a program are discussed and the idea is particularly applied to syntax error messages for LR(1) parsers. Many of the ideas used in pattern recognition are applicable to FIDO.

Important features of the system developed are

- It can be used in a number of applications.
- It is simple.
- It makes reasonable demands on computer resources.
- It is easy to use.

Examples of FIDO in use are given.

CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The situation which we examine here is that of a program in which some event requires the selection and output of one of a set of messages. There are many examples of this type. The one that we examine in most detail is the output of error messages from language processors. In this case the output is triggered by the processor's detecting an error. A large amount of work is done on language specification and language processor design, but the error messages given are often unsatisfactory. The reader will probably be able to think of many examples of poor error messages. One example from the B6700 FORTRAN compiler is given in Fig. 1.1 — although an experienced programmer might be able to understand why that message came to be given, the message itself is of no help in identifying the error.

There may be a number of reasons why error messages are so poor. For example, if the language processor uses a syntax directed translator it is difficult to determine the reason for the error. Other reasons could be that:

a. Until a system is completed, the programmer may find it difficult to foresee what errors a user can make.

b. Once a language processor has been developed, it may be difficult to add or change error messages.

c. There is little incentive for a programmer to put time and effort into developing good error messages. If the action of the processor is incorrect in the sense that it fails to process syntactically correct input or it fails to detect syntax errors, the programmer must correct this to make the processor work. However the system will still work no matter how bad the error messages are.

Under normal conditions, little can be done about (a) above, though the use of a table of messages can relieve problem (b). The ideal would probably be some easy and natural way to build up and progressively improve the set of error messages after the language processor is working.

We propose here an adaptive system which can be taught error messages. This system, which we call FIDO (From Input Derive Output), can be used interactively to develop or change error messages.

So far we have discussed the special case of error messages for language processors. However, our idea is applicable to any situation in which:

a. There is an event which indicates that a message is to be output. We will call this the event which triggers FIDO.

b. FIDO can examine enough information from the system, to decide on the particular message to be output.

c. The system can be used interactively, at least for the development of output messages. Fig. 1.2 illustrates this situation.

```
                                                                    START OF SEGMENT 002
      DIMENSION AN(10)                                          C   002:00001:0
      DO 20 I = 1,10                                            C   002:00001:0
  20  AM(I)=I                                                   C   002:00011:0
          *
WARNING:")" LABEL NOT ALLOWED ON THIS STATEMENT - LABEL IGNORED                    (073)

WARNING:STATEMENT FUNCTION DEFINITION OCCURS AFTER THE FIRST EXECUTABLE STATEMENT IN THE
        SUBPROGRAM                                                                 (195)

      STOP                                                      C   002:00021:0
      END                                                       C   002:00025:5
ERROR # 1:THE STATEMENT LABEL "20" WAS REFERENCED, BUT WAS NOT FOUND IN THIS SUBPROGRAM    (019)

                                                                    SEGMENT 002 IS 0009 LONG
```

Fig. 1.1 Example of Poor Error Messages

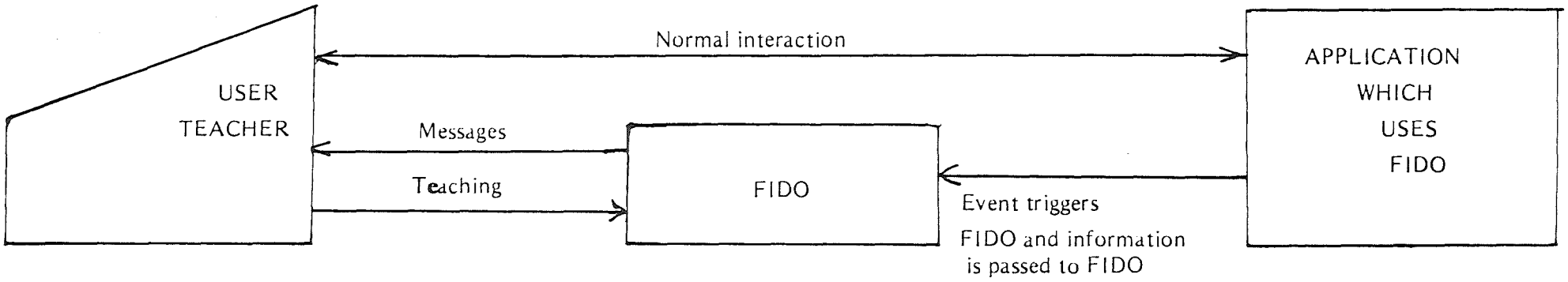


Fig. 1.2 The relationship between user, application, and FIDO

In this thesis we examine the possibility of implementing such an adaptive system, and describe a FIDO which performs adequately in test situations.

CHAPTER II

Applications for FIDO

In this chapter we briefly examine some applications for which a system such as FIDO might be used. We then discuss in more detail one application: that of giving syntax error messages from LR(1) parsers. It is this application that we use as the major example throughout this thesis.

2.1 Possible Applications for FIDO

The applications described below conform, at least in general outline, to the model illustrated in Fig. 1.1.

a. Computer Assisted Instruction

Many applications might be found in the field of CAI (Computer Assisted Instruction) for a program such as FIDO. Two examples are:

1. Eland (1975) describes a program to guide students in choosing which CAI course they should do. The student types an English question and the computer system informs the student what lessons can be done. In this system, the event which triggers action is the student's request. The information available for choosing a message is the student's request and some information on the lessons. The output message is the answer to the student's question.
2. Starkweather (1969) proposes a system which could be used to develop a student's ability to carry on a conversation in a limited subject area. Similar systems have been based on ELIZA (Weizenbaum, 1966). In such cases, the information available to a program such as FIDO, if one were to be used here, would be the conversation so far. The output messages are the computer's side of the dialogue.

b. Information Systems

A system such as FIDO might be used in developing a helpful information system, if it could learn to give appropriate information as a result of a request in English. Two possible examples are given below:

1. There have been many attempts to develop and improve document retrieval systems, among them those described by Salton (1971) and Yu (1977). The event triggering action is the request for the document. The information available to FIDO is the request and some information about the document. The output message is the document's name.
2. Natural English question-answer systems appear to be a good way for an untrained operator to get information from a computer system. EIDO might be of use in building up dialogues of this sort.

c. Other Interactive Systems

Two quite different examples are:

1. Tops 20 (DEC 1976) allows a terminal user to type a "?" and the system tells him what he can type next. Here the input to FIDO would be what the user typed before the question mark and the output message would tell the user what he can type next.
2. Johnson (1969) suggested that command languages for interactive graphics could be implemented using a syntax directed translator. Unfortunately, specifying error messages for these is never easy. This situation is similar to the ones above. The event which requires the output of a message is the breakdown of the parse. The information on which to base the selection of the message is the user's input command and the state of the parser. The output message is the error message. We will examine this example in more detail in the next section.

2.2 Syntax Error Messages for LR (1) Parsers

The syntax directed translater we consider here uses an LR (1) parser. This parser is good for interactive work because it can detect the first character after a syntax error has definitely occurred (Aho 1972). The user can therefore be informed of the error before much wasteful input has been given. The LR (1) parser, which is a special case of the LR (k) parser (Aho 1972), proceeds by shifting symbols from the input onto a stack. When the complete right hand side of a production is on the stack, it is replaced by the nonterminal symbol defined by the production.

The parser is controlled by a set of tables. Each row of the matrix in Fig. 2.1 is such a table. A table defines two functions f, g. If u is an input symbol, then for table T_1, f (u) returns that value found in row T_1, column u of the "parse action" part of the matrix. Similarly g(u) returns the value found in the "go to" part.

The state of the parser is represented by the triple (s, w, π) where s is the stack, w is the unparsed portion of the input, and π is the output list. The top item on the stack is always a table designator, and the table that it designates is said to control the stack. The initial state of the parser is: s contains only T_0

w is the whole input string

π is empty.

At each subsequent stage, if u is the leftmost symbol in w and T_i is the table controlling the stack, then, for T_i, f(u) will return a value that indicates one of four actions:

i. S, representing 'shift': i.e. u is deleted from the beginning of w, u is pushed on to s, g(u) is pushed on to s.

ii. j, an integer, representing 'reduce by rule j': i.e. if production j is of the form $W \rightarrow \alpha$ then $2|\alpha|$ symbols are popped off s. A new table, T_n, is revealed on the top of the stack; W is pushed on to s; g(W) for T_n is pushed on to s; j is appended to π

iii. X, representing 'error': i.e. the parse has failed and an error message should be given.

iv. A, representing 'accept': i.e. the parse has been successful.

For an example, see Fig. 2.1(c).

Most of the work on LR(k) parser errors has been concerned with error recovery (e.g. Poonen, 1977). That is, after an error has occurred the parser should continue checking the remainder of the input string. In interactive systems, error recovery is not so difficult, as the user himself can correct the error. He must, however, know what the mistake is. Therefore, clear error messages are important, but there appears to have been very little work done on

(0) S $\rightarrow$ S
(1) S $\rightarrow$ SaSb
(2) S $\rightarrow$ e

(a) A sample grammar

	Parsing action			Goto		
	a	b	e	S	a	b
T_o	2	X	2	T_1	X	X
T_1	S	X	A	X	T_2	X
T_2	2	2	X	T_3	X	X
T_3	S	S	X	X	T_4	T_5
T_4	2	2	X	T_6	X	X
T_5	1	X	1	X	X	X
T_6	S	S	X	X	T_4	T_1
T_1	1	1	X	X	X	X

Legend
i = reduce using production i
S = shift
A = accept
X = error

(b) LR (1) tables for the sample grammar

(To, ab, e)
($ToST_1$, ab,2)
($ToST_1 eT_2$,b,2)
($ToST_1 aT_2ST_3$,b,22)
($ToST_1aT_2ST_3bT_s$,e,22)
($ToST_1$,e,122)
(ACCEPT,122)

(c) States of an LR(k) parser passing the string 'ab'.

Fig. 2.1 Example of an LR(1) parser.

giving good syntax error messages from LR(k) parsers.

Freeth (1978) has shown one way of specifying error messages for LR(k) parsers. His method requires the specification of error messages in the grammar. When the LR(k) tables are generated, message indicators are inserted into the LR(k) tables. When the parser breaks down, a special mechanism in the parser looks up the error messages and outputs them.
For example, a production from a grammar for BASIC is:

⟨STATEMENT⟩ ::=LET ⟨VARIABLE⟩ = ⟨ARITHMETIC EXPRESSION⟩

Possible error messages relating to this production might be:

1. ILLEGAL STATEMENT START
2. INCORRECT SPELLING OF LET
3. ASSIGNMENT OPERATOR "=" REQUIRED BETWEEN VARIABLE AND ARITHMETIC EXPRESSION

The error numbers are placed into the production following a special marker ("). e.g.:

<STATEMENTS> "1 ::=LET"2 <VARIABLE> ="3<EXPRESSION>

Thus if the user typed "LET", message I would be given and if "LET=" was typed, message 3 would be given. An example of output from Freeth's system is given in Fig. 2.2.

This method of describing syntax error messages in the grammar seems to be successful. However, there are several reasons why a programmer implementing a command language could prefer not to be concerned with error messages until after the implementation is almost complete:

a. Having to specify error messages will distract the programmer from his major task.
b. The programmer will have difficulty in predicting errors the user will make.
c. There is very little motivation for a programmer to create good error messages and to keep them up to date.
d. The programmer may find it difficult to decide where in the grammar an error message should be marked.
e. If the programmer makes an error in the placement of an error marker he may have to generate the tables again.

A learning system such as FIDO may overcome these problems.

```
MINIBASIC PARSER


010 REM PROGRAM TO SHOW SOME OF THE ERROR MESSAGES.
* SYNTAX ERROR
 ILLEGAL LETTER NOT A MEMBER OF THE ALPHABET.
 EXPECTING TERMINATION OF STATEMENT BY <CR>.

010 REM PROGRAM TO SHOW SOME OF THE ERROR MESSAGES <CR>

020 LET AA
* SYNTAX ERROR
 A VARIABLE CAN ONLY BE A <LETTER> OR A <LETTER><DIGIT>.
 ASSIGNMENT OPERATOR "=" EXPECTED BETWEEN VARIABLE AND ARITH EXPRESSION

020 LET A1= 3 * 4,
* SYNTAX ERROR
 REQUIRES AN ARITH OPERATOR +-/*@ BETWEEN OPERANDS
 EXPECTING TERMINATION OF STATEMENT BY <CR>.

020 LET A1= 3 * 4. 0 * 2 <CR>

030 LET B  = *
* SYNTAX ERROR
 ARITH EXPRESSION IS EXPECTED.

030 LET B  = + A1 * 6/7 <CR>

040 LET C  = A1* C + D )
* SYNTAX ERROR
 REQUIRES AN ARITH OPERATOR +-/*@ BETWEEN OPERANDS
 EXPECTING TERMINATION OF STATEMENT BY <CR>.

040 LET C  = A1* C + D <CR>

050 LET F  = (C*A )/(E - C 9
* SYNTAX ERROR
 REQUIRES AN ARITH OPERATOR +-/*@ BETWEEN OPERANDS
 BRACKETED ARITH EXPRESSION REQUIRES RIGHT BRACKET.

050 LET F  = (C*A )/(E - C  )<CR>

060 LET G  = A1 * C + D  +)
* SYNTAX ERROR
 OPERAND OR BRACKETED ARITH EXPRESSION EXPECTED.

060 LET G  = A1 * C + D  + C + 3E4 <CR>

070 IF A ><
* SYNTAX ERROR
 ARITH EXPRESSION IS EXPECTED.

070 IF A >= B GOO
* SYNTAX ERROR
 INCORRECT SPELLING OF GOTO.

070 IF A >= B GOTS
* SYNTAX ERROR
 INCORRECT SPELLING OF GOTO.
```

Fig. 2.2 Example of interaction with Freeth's System

CHAPTER III

A REVIEW OF POSSIBLE TECHNIQUES

Our use of an interactive system for 'learning' error messages is, as far as we can tell, unique. Certainly, we have found no reference describing an application similar to our own. Adaptive systems have, of course, been developed for many purposes, and a wide variety of techniques has been employed. Our aim is to determine which of these techniques might be usefully adapted for our purpose. Different approaches arise from varied objectives. For example, some people are interested in trying to simulate a brain, that is, to make a computer behave in a way that is equivalent to the way a person would behave. PURR-PUSS (Andreae, 1976) is an example of such a system. For FIDO, we have no such constraint, and might find more of interest in systems designed (by any method) simply to acquire knowledge in certain areas. The program known as SAD SAM (Lindsay, 1973), for example, acquires knowledge and answers questions about kindred relationships. Another example is SIR (Raphael, 1968), designed to learn relationships between various objects and answer questions on these relationships.

However, FIDO is probably more like a pattern recognition system. In each case the system is presented with some information and is required to select one of a number of alternatives. In the case of FIDO this is the message number. In the case of pattern recognition it is the pattern class.

We therefore concentrate our discussion on techniques used for pattern recognition.

3.1 Review of Pattern Recognition

Pattern recognition systems can be broadly divided into two kinds – trainable (adaptive) and untrainable (non-adaptive). A common example of non-adaptive pattern recognition can be found in systems which read Magnetic Ink Character Recognition Code (MICR) as printed on bank cheques (Spencer, 1968, Appendix Q). Another interesting pattern recognition system is the well known ELIZA program (Weizenbaum, 1966). Fig. 3.1 from Tou, (1974) page 6 illustrates some of the uses of pattern recognition, and the reader is referred to Fu (1976) for a review of pattern recognition literature.

Task of Classification	Input Data	Output Response
Character recognition	Optical signals or strokes	Name of character
Speech recognition	Acoustic waveforms	Name of word
Speaker recognition	Voice	Name of speaker
Weather prediction	Weather maps	Weather forecast
Medical diagnosis	Symptoms	Disease
Stock market prediction	Financial news and charts	Predicted market ups and downs

Fig. 3.1 Some uses for Pattern Recognition Systems

Trainable pattern recognition systems are of course more relevant to our problem. Tou (1974) identifies two broad types of trainable pattern recognition systems. One is the unsupervised system (sometimes called learning without a teacher). Here the system is given samples of patterns but is not told to which class they belong. The system is required to find a number of classes into which these patterns fall. This sort of system would normally use techniques such as cluster analysis (Anderberg, 1973). An example of where unsupervised learning might be used is to group hospital patients with unknown diseases.

FIDO's learning is supervised. That is, for each pattern presented during training, the system is told which pattern class it belongs to or, in the case of FIDO, which message should be output. A good example of a trainable pattern system is the voice recognition systems now being advertised.

Most pattern recognition systems require the input data to be preprocessed. Usually this results in a pattern vector. For example, preprocessing in a character recognition system may result in a vector where each value represents the presence or absence of ink on a particular spot on the paper.

Preprocessing is sometimes referred to as feature extraction. Most text books on pattern recognition (e.g. Tou, 1974 or Uhr, 1973) have sections on feature extraction. Once features have been extracted, a number of techniques can be used by pattern recognition systems. These techniques usually fall into one of the following categories — syntactic, heuristic or mathematical. Each of these techniques is discussed in the following sections.

3.2 Syntactic Methods of Pattern Recognition

The process of building grammars from samples of strings in the language is called grammatical inference. Much work has been done on this (e.g. Biermann, 1972a, 1972b, 1972c; Fu, 1975).

If we consider only character string input to FIDO, one possible structure for FIDO's memory would be a phrase structure grammar (e.g. context free). For each output message FIDO would have a formal grammar which would describe the set of input strings requiring that message to be given. At first the idea of using a formal grammar as the memory of FIDO appears attractive when dealing with LR(k) syntax error messages. In this application each grammar held by FIDO would be a modification of the grammar used to generate the parser. For example, suppose we are parsing BASIC. The user types:

```
10 LET A B
```

At this point the parse breaks down and FIDO is called. FIDO will examine the input string and must give a message such as

```
'=' REQUIRED IN LET STATEMENT
```

As part of FIDO's memory there must be a grammar G which describes a language L(G), such that when an input to FIDO is in L(G) the above message must be given.
The grammar G might be

<start> : : = LET <var> <inval>

<inval> : : = {any character except '='}

In teaching mode, FIDO would use grammatical inference for the memory update mechanism.

We suggest that the idea of grammatical inference is not suitable for FIDO because it would be too slow. If only the input string to the parser is used by FIDO then a wealth of information is lost; for example, the stack of the parser can often pinpoint very closely the error. Retention of some of this information would result in faster learning. Another reason why a grammatical inference system would learn slowly is that the grammars to be inferred might be rather complex (the reader might try to write a grammar to match all strings requiring the message 'mismatched parentheses' to be given from a BASIC parser).

3.3 Heuristic Techniques

Slagle (1971) describes an heuristic as a rule-of-thumb strategy, method or trick, used to improve efficiency of a system which tries to discover the solution to a complex problem. It is impossible to give an overview of heuristic techniques because they cover such a wide range of ideas. The distinguishing feature of these techniques is that they work because some programmer had an insight or idea which he built into a program.
This approach might be referred to as an ad hoc approach as it is specific to one application. Lindsay (1973) defends the use of ad hoc methods in artificial intelligence. Not only can an ad hoc system lead to an easily implemented and practical system, but it can also aid understanding of how a more general system might be developed.

We have introduced FIDO as a system intended to work in a number of situations. However, it could be that some of the differences in inputs would make it very difficult to have such a general system. Therefore it might be suggested that to build a system to learn LR(1) syntax error messages, only that application should be considered.

We have already described Freeth's system which handles LR(1) syntax error messages in a non-learning way (Section 2.2). It would seem reasonable that FIDO should try to employ heuristic techniques to build error message numbers into LR(k) tables so that the tables end up looking like those employed by Freeth. An objection to the above method is that Freeth's system modifies LR(k) tables to suit the giving of good error messages. This, we believe, can be done only when the tables are generated, a process that is very costly on computer resources. (To generate tables for a subset of BASIC, using Freeth's program, took 183 seconds of CPU time on a B6700 computer). As this would have to be done whenever FIDO's memory was updated this method must be excluded on the grounds of response time and use of computer resources.

There are many possible ad hoc procedures which could be applied to this problem. However, if possible, it would be desirable to develop a general system that could be used in many applications. Our final system will contain one ad hoc module, but most of FIDO is more general.

3.4 Mathematical Techniques

The mathematical approach to pattern recognition uses classification rules based on a mathematical framework. With most of these techniques, it is assumed that a vector of real numbers $X = (X_1, X_2 \ldots X_n)$ has been extracted from the input data by a pre-processor as described earlier. This vector describes a point in n-space. The job of any classifier is to map points in n-space on to pattern classes.

One technique used to accomplish this mapping is referred to as 'the nearest neighbour rule' (Cover, 1967). In its simplest form this technique stores all pattern vectors and their classifications. When a new pattern vector is encountered the distance through hyperspace from it to each other vector is calculated. The new vector is classified in the same class as the closest old vector. Some variations on this technique consider the classification of the k nearest neighbours, where k is some constant. We feel that nearest neighbour techniques would tend to use too much processor time to be very practical.

Most mathematical techniques attempt to define functions which determine surfaces in hyperspace. These surfaces separate pattern classes. In the simplest form of the nearest neighbour system, these surfaces are half-way between vectors of one classification and neighbouring vectors of another classification. Fig. 3.2 illustrates this in 2-space. The a's represent pattern vectors from Class A and the b's from Class B. The surface separating patterns of Class A from Class B is drawn.

Another technique attempts to discover the formulae of the hyper-surfaces which separate pattern classes. Arkadev (1967) describes some methods of doing this. The simplest type of formula describing a hyperplane is a linear one. That is, let $W = (W_1, W_2 \ldots W_n)$ be a vector which we will call a weight vector. Let $W.X = C$ (where C is some constant) be the formula of some hyperplane which separates the pattern class H_1 and H_2. The decision of whether a particular pattern X belongs to class H_1 or H_2 can be made as follows:

If $W.X < C$ then H_1

If $W.X \geq C$ then H_2

The task of an adaptive pattern recognition systen would be to derive the weight vector W and the constant C.

Where there are more than two pattern classes, a weight vector for each class can be used. These we will call W^1, W^2, $\ldots W^m$ where there are m classes. We now decide that X belongs to class H_i if:

$W^i.X < W^j.X$ for all $j \neq i$.

Such a system is discussed in Ho (1968). This system we shall refer to as a 'Linear decision function'. As will be seen in Chapter 5 our FIDO is based on a simple form of linear decision functions.

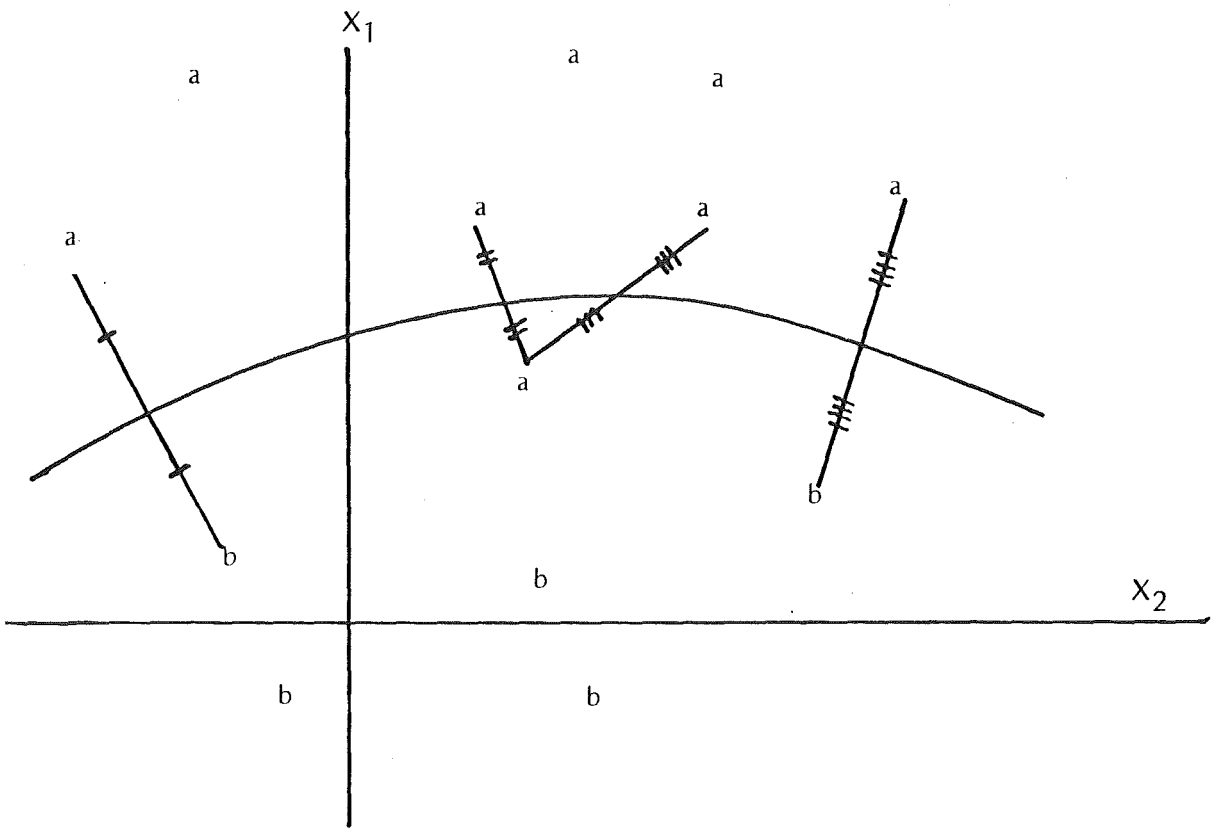


Fig. 3.2 Hypersurface defined by nearest neighbour rule.

CHAPTER IV

BACKGROUND IDEAS

In this chapter we discuss some of the ideas we consider important when designing a FIDO. The first two sections describe FIDO from different points of view. A formal description is useful to check that our model of FIDO is complete. Secion 4.2 describes how FIDO will appear to a user.

An important consideration when designing FIDO was the type of input data FIDO had to base its decisions on. Possible types of input are discussed in Section 4.4.
The last section in this chapter gives a skeleton outline which would be appropriate for any method of implementing a FIDO program.

4.1 Formal description of FIDO

When in operating mode FIDO is formally described by the triple $\emptyset = (M, I, \delta)$
where M is the set of messages FIDO can output.

I is the set of all possible inputs to FIDO.

δ is a mapping such that δ: $I \rightarrow M$.

i.e. for any input to FIDO, FIDO must give exactly one message.

When in training mode FIDO may be changed in two ways:

i. More output messages may be added.

ii. The mapping function may be changed.

To do this we will have a function G.

$\emptyset' = G\ (\emptyset,\ i,\ c)$

where $\emptyset = (M, I, G)$ as it was before this teaching episode.*

$\emptyset'$ is the updated FIDO $(M',\ I,\ \delta')$

$i \in I$ is the input used for this teaching episode.

$c \in M'$ is the correct message to be given in response to i.

i.e. While in training mode FIDO is modified as a result of the teaching.

4.2 User's View of FIDO

FIDO will function in one of two modes:

a. *Operating Mode* will be the normal mode. When it is required that a message be given FIDO will be called. FIDO will examine information passed to it from the calling system and output the appropriate message, one of a set previously given in training mode.

*: A teaching episode for FIDO comprises three events:

i. the user types some input.

ii. FIDO issues some message in response.

iii. the user responds with the correct message as explained in more detail in Section 4.2.

b. *Training Mode* can be used during the initial development of the set of messages, or if a user is dissatisfied with an existing set. In this mode, after it is called, FIDO will give a message as in operating mode, but will then wait for feedback from the 'teacher'. The aim of the teacher is to improve the quality of the messages and, as a result of his reactions, FIDO should adjust its internal states in such a way that its performance will improve.

The teacher should be able to give FIDO a number of different responses:

1. *Yes.* This response indicates that the message was acceptable. In the present implementation, to give this response the teacher types 'Yes' or 'Y'.
2. *No, try again.* This response shows that the message was not acceptable and that FIDO should try for another. To give this response the teacher types 'No' or 'N'.
3. *No, the answer is message x,* where x is an integer. This response indicates to FIDO that the message numbered 'x' (of the set of messages held by FIDO) should have been given. To give this response the teacher types the integer x.
4. *No, the answer is . . .* This response allows the teacher to provide a new message. To give this response the teacher types a double quote followed by the message.

In addition, FIDO requires utility functions to enable users to correct or change messages. For examples of teaching lessons refer to the appendices.

4.3 Performance Criteria

Before deciding on the form of a computer model of FIDO we must consider the criteria on which FIDO will be evaluated. Some of these will be critical in deciding upon how FIDO will be implemented. Others will mainly affect the input and output modules.

Wherever possible, the criterion should be quantitative, even though when dealing with the psychological requirements of the user it is very difficult to lay down fixed measures of required performance. We believe that important criteria for FIDO are:

a. Easy to Learn to Use

For LR(k) syntax error messages FIDO is intended to be used near the end of what could be a long and tedious development process. As stated in Chapter II, FIDO should be a simple and natural way of developing error messages. We suggest that a programmer should be able to learn to use FIDO in about 15 minutes.

b. Natural to Use

The form of the dialogue with FIDO must be such that the user is not frustrated by it. It would be desirable that the teacher could respond to FIDO in much the same way that he would respond to a person. This would help to make FIDO easy and natural to use.

c. Good Response Time

Martin (1973, Chapter 18) defines response time as 'the interval between the operator pressing the last key in the input operation and the terminal displaying the first character of the response'. When handling LR(k) syntax errors, this time is the time taken for the parser to detect the error plus the time taken for FIDO to decide upon and give the error message. A response time of two seconds is considered the maximum acceptable for interactive work.

d. Learns Well

For FIDO to be acceptable it must learn in a manner which is acceptable to the teacher.

When considering how well FIDO is learning we feel two things are important:

i. It must be possible for FIDO to give the correct message in any situation; i.e. after a suitable period of training FIDO should always give the correct message. In practical implementation this period might be very long. Therefore this criterion might be reformulated thus: when FIDO gives a wrong message its performance can be improved with further training.

ii. The learning of FIDO must be fast enough to keep the teacher satisfied.

We will say that FIDO learns well enough if it can be taught a satisfactory set of messages in a significantly shorter time than it would take to develop a similar set using some other technique.

e. Economic Use of Computer Resources

It is important that, in attempting to meet the other criteria, FIDO does not make an unacceptable demand on computer resources.

4.4 The Nature of Inputs to FIDO

FIDO may use any or all of the information available to it through the input mechanism. When handling syntax error messages from an LR(1) parser FIDO has various sources of information. These are:

a. The input typed by the user up to the detection of an error.

b. Input typed by the user after the detection of the error. If the error is reported as soon as it is detected there would presumably be no further input.

c. The parse (rule numbers) already produced by the parser before the breakdown of the parse.

d. A stack of grammar items and parser states. This stack together with the look-ahead-character represents the state of the parser at the time the parse breaks down.

e. The look-ahead-character, which is the lexical item the parser was trying to match when the parse broke down.

f. Some information from the lexical analyser, such as whether a blank occurred in the lexical item being processed.

The above represent several data types, each with its own important characteristics. Some of the types are:

1. Character strings such as (a.) above.
2. Ordered data, which are a sequence of data items where the order of the items makes a difference to the meaning or interpretation of the data. Character strings are a special case of ordered data. Other examples are (c.) and (d.) above.
3. Structured data, where the interaction between data items is more complex than simple ordering. A matrix is one example of structured data and another example is a tree representing the parse so far (c.) above.
4. An isolated data item, one which has no significant relationship to other data items. The look-ahead-character (e.) above is an example.

4.5 Essential Components of a FIDO

An examination of the FIDO problem suggests that FIDO can be broken down into a number of components. Each of these components will represent a module or data structure of the computer program used to implement FIDO. We present these components here to simplify the discussion of possible methods of implementing FIDO.

We feel that FIDO must have the following components for it to behave in the way we have described.

a. An INPUT MODULE to accept input into FIDO from some other module of the computer system (for example, an LR(1) parser). This mechanism might put the input data into some standard form to make it easier for the rest of FIDO to process it.

b. A DECISION MODULE to decide upon which output messages should be given. If FIDO is looked on as a classification system, then the decision mechanism must decide to which class an input belongs.

c. The OUTPUT MODULE to give the output message to the user (or teacher).

d. An ACCEPT-FEEDBACK MODULE to receive feedback from the teacher while FIDO is in training mode. The accept-feedback module would normally get its input from the teacher's terminal, in contrast to the input module which would get its input from some other module in the system.

To enable these modules to perform their functions, certain other components are required, namely:

e. A MEMORY which can contain:
   i. Enough information to allow FIDO to decide on the correct message.
   ii. The text of the messages.

f. A MEMORY-UPDATE MODULE which would be a set of algorithms to modify memory so that the decision mechanism can give the correct answers.

The decision module, the memory-update module, and the memory itself do not interact directly with the user. It is, however, these components which will determine how successfully FIDO will learn.

Fig. 4.1 illustrates these parts.

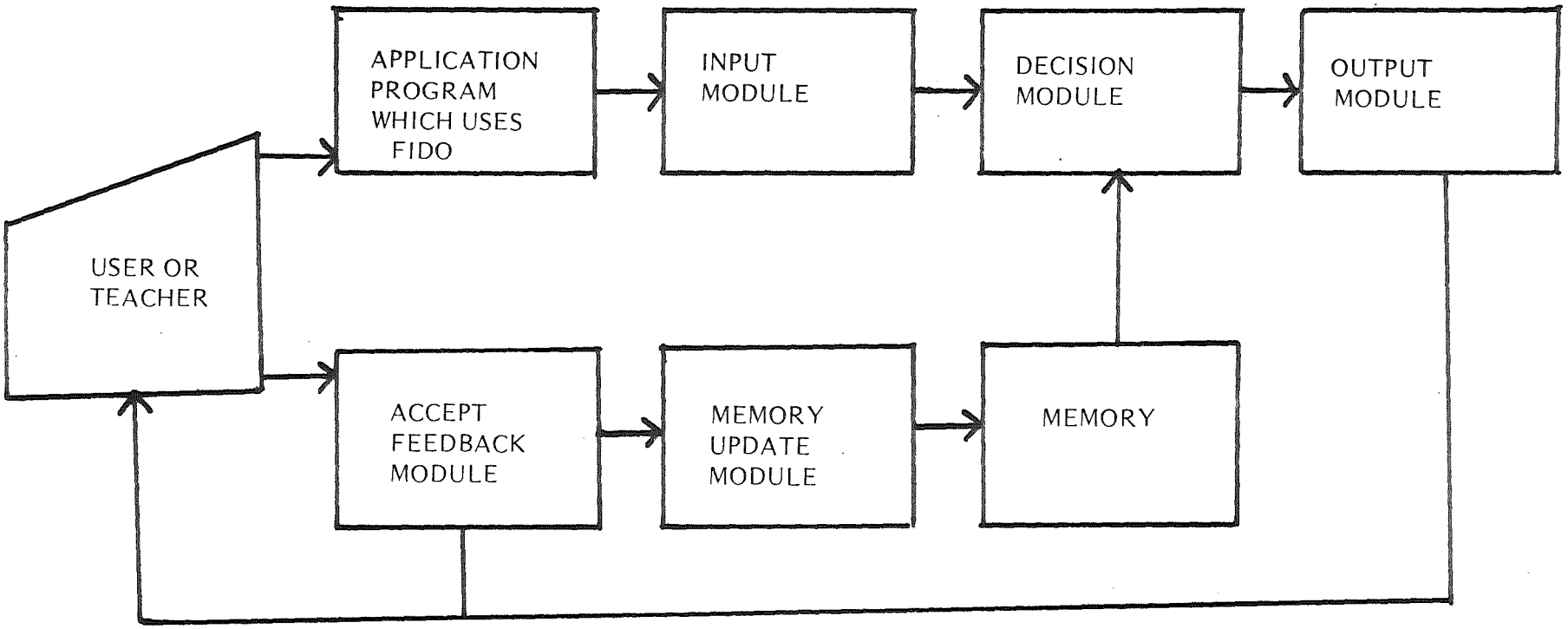


Fig. 4.1 Possible structure for any FIDO

CHAPTER V

A PRACTICAL IMPLEMENTATION

In this chapter we present our method of implementing a FIDO. We describe the basic ideas in sections 5.1 and 5.2. Section 5.3 describes some important points and additions to the basic ideas. That section also contains descriptions of two other approaches which we considered when developing FIDO. Finally, section 5.4 discusses practical considerations such as the arranging of FIDO's memory and the detail of the form of the dialogue with FIDO.

5.1 Feature Extraction

A major problem found by the author when designing FIDO was that the information on which FIDO must base its learning can be in several different forms (Sec. 4.4). A FIDO which could use only one of these forms would lose valuable data. (The reader is referred to the criticism of grammatical inference in Section 3.2). On the other hand, we believe it would be very difficult to design a FIDO which could handle information in these several different forms. The obvious solution to this dilemma is to convert all the input information into a standard form using heuristic techniques.

This conversion will be done by the input mechanism. Therefore the input mechanism is not just a collector of information, but also transforms it into a form FIDO can handle. The input mechanism must now be rewritten for each application for which FIDO is used, making FIDO less general, but reducing it from an overwhelmingly difficult problem to a practical proposition. Having decided to shift some of FIDO's work to an *ad hoc* module two questions must be answered.

1. How much of FIDO's work should be done by the input mechanism?
2. What is the standard form of information that it will output to the rest of FIDO?

The input mechanism implements feature extraction as discussed in Section 3.1. It is important that we do not make the input mechanism too hard to program. On the other hand we should recognise that the programmer has a degree of intelligence we could never give to FIDO, as he has some insight into what are the important characteristics or features of inputs. Provided that clear instructions can be given, it is reasonable to ask the programmer of the input mechanism to extract these as features and pass them on to the learning part of FIDO.

Question (2) above can now be rephrased as "What is a feature?" We decided that a feature should be a flag to indicate a characteristic of the inputs. Therefore the input mechanism gives a set of features indicating certain characteristics of the input. It should be pointed out that the order of features in the set will bear no relation to their meaning. If the order or structure of the input date is important then this is a characteristic of the input and suitable features must be created to inform FIDO of this.

We do not claim that all information can be easily translated into this form, but we do believe that such features can adequately represent a good many types of input data. For example:

a. An integer in the range from n to m can be represented by m-n+l features.
b. A specific English word can be represented by one feature.
c. A stack which will never get deeper than j and will only ever hold i values at any level can be represented by i times j features. This can be reduced if we are only interested in items near the top of the stack.
d. An arbitrary character string is harder to represent. However the programmer may have some insight into the meaning of certain substrings, and he may choose to represent this meaning as features.
e. A real number would be difficult to represent. In most of the applications we envisage, real numbers will not occur. If a programmer wished to represent real numbers he would have to break them up into ranges of values and represent each range as a feature.

When a feature is in the set output by the input mechanism we say that that feature 'occurred'. For the rest of this chapter we will talk about features rather than inputs. We will further discuss feature extraction when we give practical examples.

5.2 Bitmap FIDO

This section explains one possible FIDO. While discussing this FIDO, we will describe several other FIDO's. This FIDO has been shown to perform adequately in two applications (syntax error messages for LR(k) parsers and a question answering system). We will refer to this FIDO as Bitmap FIDO after one possible implementation technique and we will use Algol 60 notation to help describe its operation. The algorithms we give will not be the most efficient ones, but we feel that they clearly represent what we are trying to do. When FIDO is implemented, suitable programming techniques would be used to increase computer efficiency. Some of these techniques will be discussed later. When designing the algorithms given below, we assumed that each feature is represented by an integer in the range 1 to MAXMESS. The function OCCURRED (i) is true if feature i occurred, false otherwise. The memory of Bitmap FIDO is a boolean matrix (Fig. 5.1):

MEMORY [1: MAXFEAT, 1: MAXMESS].

MESSAGES

	1	2	3
1	F	F	F
2	F	F	F
3	F	F	F
4	F	F	F
5	F	F	F

MAXMESS = 3

MAXFEAT = 5

Fig. 5.1 Initial state of FIDO's memory assuming 5 features and 3 messages.

	1	2	3
1	F	T	F
2	F	T	F
3	F	F	F
4	F	T	F
5	F	F	F

F = false

T = true

Fig. 5.2 FIDO's memory after update.

To understand how Bitmap FIDO works it is best to look first at the memory update algorithm. Assume that the correct message number for some set of features is CORRECT. To update its memory, FIDO sets true MEMORY [i, CORRECT] for each i where the feature numbered i has occurred. For example, suppose that the features 1, 2 and 4 occur and the correct answer is message 2. If the memory is initially empty (i.e. all features false) the result of the memory update would be as shown in Fig. 5.2.

The Algol 60 routine to do this is:

```
for i : = 1 step 1 until MAXFEAT do
 if OCCURRED (i) then
   MEMORY [i, CORRECT] : = true ;
```

If MEMORY [i, j] is true we say that feature i indicates message j. The decision mechanism uses this matrix to decide which message to give. To do this each message is scored using a scoring algorithm and the message with the highest score is selected. If every time feature F_i occurred message M_j was the correct answer it would seem to be reasonable that the scoring algorithm should, in this situation, give feature F_i a comparatively large weight. If, however, feature F_i indicates many messages, F_i should receive little weight.

The scoring algorithm assumes a function INDICATES (FEAT) which returns the number of messages indicated by feature FEAT. (i.e. how many elements in the row FEAT of the array MEMORY are true). The scoring algorithm is:

```
comment initialize the scores;
  for i: = 1 step 1 until MAXMESS do SCORE [i] : = 0;
comment the scoring algorithm;
  for i: = 1 step 1 until MAXFEAT do
    if OCCURRED (i) then
  for j : = 1 step 1 until MAXMESS do
    if MEMORY (i,j) then
     SCORE [j] : = SCORE [j]+ 1/ INDICATES (j) ;
```

i.e. For each feature that has occurred, if that feature indicates M_j, the inverse of the number of messages that feature indicates is added to the score for M_j.

The message with the highest score is selected and output.

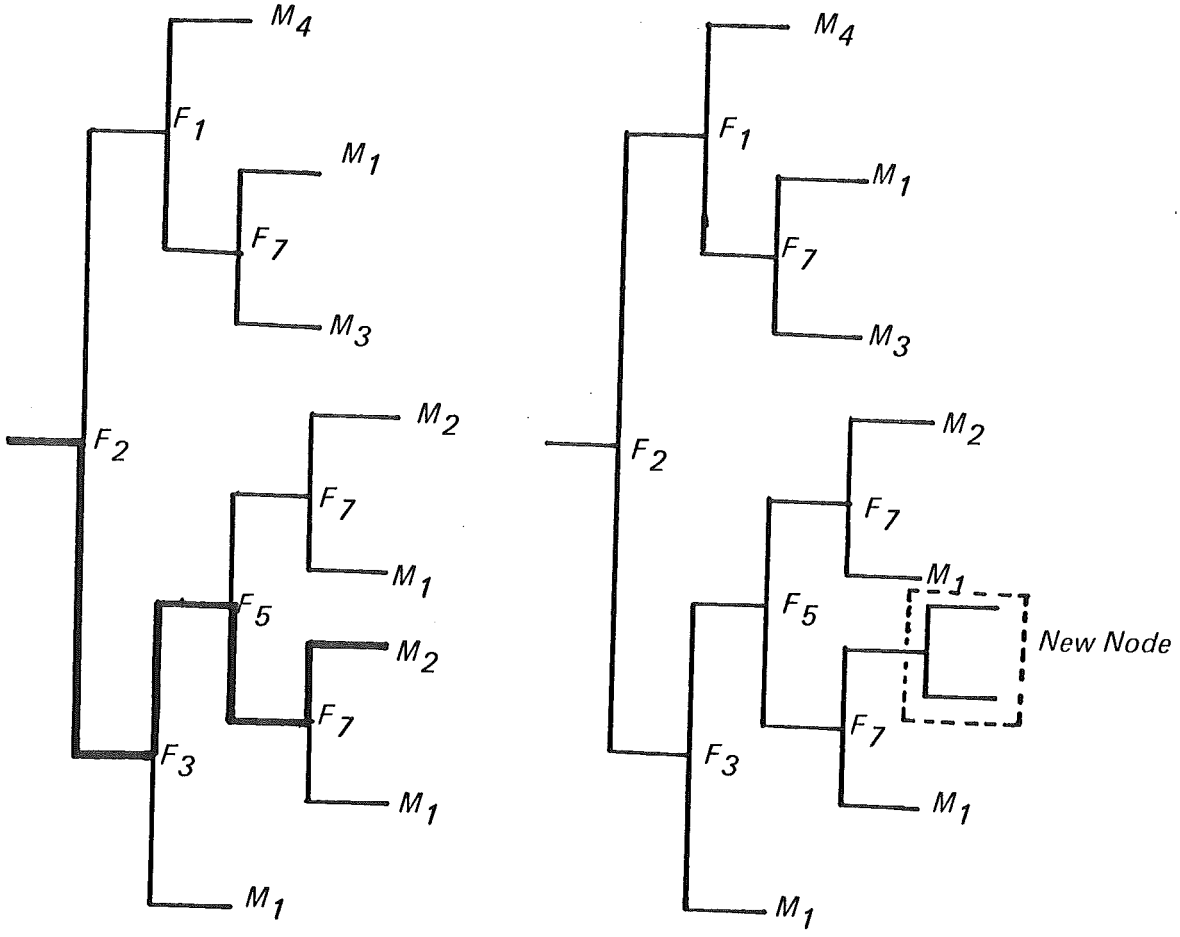
5.3 Important Characteristics of Bitmap FIDO

The algorithms given in 5.2 describe the basis of Bitmap FIDO, but there are some additional concepts needed to make FIDO practical. We will describe these concepts when the problems which they are designed to overcome arise in the discussion. Some important characteristics of Bitmap FIDO's design and performance are:

a. Bitmap FIDO Learns Fast

Probably the most important property of Bitmap FIDO is the speed at which it learns, provided that the problem is one that FIDO can handle. We will discuss later the situations FIDO cannot handle. One reason that FIDO learns quickly is that it uses, as a basis for learning, all the features which have occurred. This allows the features which are most important to be reflected quickly in the result of the scoring algorithm.

This speed of learning can be contrasted with another FIDO developed by the author. We will refer to this FIDO as Tree FIDO. Tree FIDO is capable of learning the correct message for any possible combination of features i.e. Tree FIDO's learning will be complete. The memory of Tree FIDO is a tree. At each node in the tree a feature is stored. The decision mechanism starts at the root of the tree and works its way up to a leaf. At each node the decision mechanism tests the feature which is stored there to see if it has occurred. If it has occurred one branch is taken (we call this the 'Up branch') and if the feature has not occurred the other branch is taken ('down'). When a leaf is reached the message stored on that leaf is output. For example; suppose features F_1, F_3 and F_7 occur and the memory is the tree in Fig. 5.3a. The path taken through the tree is marked with the heavy line and message M_2 is given.



(a) A path through Tree FIDO's Memory.

(b) Addition of a new node

Fig. 5.3 Example of Tree FIDO's Memory.

The memory update mechanism must modify the tree so that Tree FIDO gives the correct answer. A number of methods have been investigated. The simplest method is to add a node at the leaf where a wrong answer is given. The new node would test a feature which has occurred but was not tested on the way up the tree. (See Fig. 5.3b).
A major difference between Bitmap FIDO and Tree FIDO is that while Bitmap FIDO updates a portion of memory relating to a number of features, Tree FIDO updates only one node on the tree. This means that one update of Bitmap-FIDO's memory affects more situations and therefore its learning is faster than that of Tree FIDO.

b. Bitmap FIDO has economic memory usage

An apparently attractive alternative for FIDO's memory is an integer matrix which holds in MEMORY [i,j]the number of times feature F_i has occurred when message M_j has been the correct answer. The memory update algorithm would be:

```
for i : = 1 step1 until MAXFEAT do
  if OCCURRED then MEMORY [i, CORRECT]:= MEMORY [i, CORRECT]+ 1;
```

The scoring algorithm would be:

```
for i : = 1 step 1 until MAXFEAT do
  if OCCURRED (i) then
    for j: = 1 step 1 until MAXMESS do
      if MEMORY [i, j]≠ 0 then
        SCORE (j) : = SCORE (j) + 1/TOTAL(i);
```

TOTAL (i) is an integer function which sums up row i of memory (i.e. the row corresponding to feature F_i).

We shall refer to this method as Count FIDO. The author feels that this method is in several ways inferior to Birmap FIDO. The one situation in which Count FIDO may have an advantage over Bitmap FIDO is when some previous teaching must be forgotten. (i.e. if a teacher makes a mistake or a new teacher decides to teach a new message for a situation that FIDO has already been taught so that previous teaching must be forgotten.)

The most obvious disadvantage of Count FIDO is the amount of memory required. If Bitmap FIDO uses one bit of memory for each boolean value and Count FIDO uses 8 bits (256 possible values) then Count FIDO requires 8 times as much memory as Bitmap FIDO. This could easily force FIDO's memory out of main memory of the computer and thus slow FIDO down below acceptable performance levels.

c. Bitmap FIDO is Economic on Processor Time

The binary type features in FIDO are very simple. For this reason we would expect a system using FIDO to have a large number of these. However, we would only expect a small proportion of them to occur at any one time. Since the scoring algorithm needs to look at memory for only those features which have occurred, scoring will be fast. Similarly, the memory update algorithms will need to set only a small number of bits in memory. The calculations involved are also very simple. If a count of the number of messages each feature indicates is kept, the scoring algorithms need only do one division for each feature that occurred and one addition for each message indicated by each feature which has occurred. Therefore the demand in processor time is minimal.

d. Bitmap FIDO Learns Infrequent Situations

Suppose that there is a rarely occurring situation which requires a message of its own. It is likely that such a situation might be taught only once by a teacher. Often this type of message is important. For example, it may be the unusual error situation in a program which is hard to recognise. Bitmap FIDO does not hold the frequency of a message taught to be important. Provided that a situation has been taught once, FIDO has learnt it.

Compare this with Count FIDO, described in (b). Suppose that a teacher first teaches FIDO that when features F_1, F_3, and F_5 occur together message M_2 is the correct answer. Count FIDO's memory would be as in Fig. 5.4a and Bitmap FIDO's as in Fig. 5.4b.

Suppose the teacher next trains FIDO that F_3 and any other features except feature F_1 gives message M_1, Count FIDO's memory would be as shown in Fig. 5.4c and Bitmap FIDO's would be as shown in Fig. 5.4d. Suppose we now get the F_1, F_3 and F_5 input. If the respective scoring algorithms are applied Count FIDO selects M_1 and Bitmap FIDO selects M_2 (i.e. Bitmap FIDO gets the correct answer.) This property is also important if a teacher wants to reteach FIDO. It would be unacceptable for a teacher to wait while the counts for his teaching slowly overtake those for the previous teaching.

(a)

	M_1	M_2
F_1	0	2
F_2	0	0
F_3	0	2
F_4	0	0
F_5	0	2

(b)

	M_1	M_2
F_1	F	T
F_2	F	F
F_3	F	T
F_4	F	F
F_5	F	T

(c)

	M_1	M_2
F_1	0	2
F_2	5	0
F_3	10	2
F_4	6	0
F_5	5	2

(d)

	M_1	M_2
F_1	F	T
F_2	T	F
F_3	T	T
F_4	T	F
F_5	T	T

Fig. 5.4 Comparison of Bitmap and Count FIDO.

e. Bitmap FIDO Cannot Learn Some Things

Suppose that when features F_1, F_2 and F_3 occur together, message M_1 is required, and when F_1 and F_2 occur together M_2 is required. Fig. 5.5a shows what the memory of Bitmap FIDO would be. Now, if F_1 and F_2 occur, M_1 and M_2 will get the same score and FIDO will not be able to discriminate between the two messages. Three approaches to solving this problem are:

1. Output both messages. This should be considered as only a last resort. It is desirable to give a user a specific message rather than give him a choice.
2. Ask the writer of the input mechanism to provide more features. In our experience, this problem occurs rather often and it is difficult to find features to solve the problem. We therefore feel that a better solution is:
3. Make some addition for FIDO to handle this situation. In doing so, we do not want to force FIDO to look at large portions of memory. This rules out trying to find the significance of a particular feature not occurring.

A simple addition to Bitmap FIDO which makes it possible for FIDO to work for the above example is to have FIDO check to see if there are two messages with the same highest score. If there are, FIDO generates an extra feature which we will call a 'pseudo feature'. There would be a pseudo feature for each pair of messages which have in the past had equal scores. Fig. 5.5b shows what FIDO's memory would look like for the above example.

The adding of these pseudo features does not completely solve the problem of things FIDO cannot learn. For example, suppose F_1 by itself gives M_1, and F_1, F_2 and F_3 gives M_1. FIDO's memory would be as in Fig. 5.5c. If this type of situation occurs, additions to feature extraction would be required.

f. Bitmap FIDO Has a Problem with Forgetting

Bitmap FIDO as described fails if a teacher makes a mistake while teaching FIDO or if it is decided to teach a new message in a situation which has already been taught. The problem with the algorithms so far described is that once a boolean in the memory becomes true it will never be reset. We shall refer to this problem as 'the forgetting problem' and the process of turning memory values to false as 'forgetting'. Forgetting should happen when FIDO has given an answer which is incorrect and updating memory would not result in the correct answer being given if FIDO was called with exactly the same features.

The need for forgetting would be reduced by the pseudo features described in (e).

(a)

	M_1	M_2
F_1	T	T
F_2	T	T
F_3	T	F

(b)

	M_1	M_2
F_1	T	T
F_2	T	T
F_3	T	F
P_1	F	T

(c)

	M_1	M_2
F_1	T	T
F_2	T	T
F_3	T	F
P_1	T	T

Fig. 5.5 Examples of Pseudo Features.

Suppose that we have trained FIDO that F_1 and F_2 gives M_1, F_1 and F_3 gives M_2, and F_1 and F_4 also gives M_2. The memory would be as shown in Fig. 5.6a. If the teacher now tries to retrain FIDO so that features F_1 and F_3 together give message M_3, then whenever F_1 and F_3 occur messages M_2 and M_3 will get the same score. A pseudo feature P_1 would be generated by FIDO and the updated memory would be as shown in Fig. 5.6b. FIDO has now learnt the new situation and effectively forgotten the old one.

(a)

	M_1	M_2	M_3
F_1	T	T	F
F_2	T	F	F
F_3	F	T	F
F_4	F	T	F

(b)

	M_1	M_2	M_3
F_1	T	T	T
F_2	T	F	T
F_3	F	T	T
F_4	F	F	F
P_1	F	F	T

Fig. 5.6 How pseudo features can help correct teachers' errors.

However, it is hoped that FIDO will be capable of being taught by several different teachers and over a period of time. Each teacher's idea of what message a particular situation should give might be different, or a teacher might decide that a message is too general and should be split into several parts. As a result a situation may be retaught several times in the life of a system. This could well mean that adding pseudo features would not be adequate to handle new teaching. In this case a specific forgetting algorithm to turn values in FIDO's memory to false is required.

The decision to be made in the design of the forgetting algorithm is: What elements in FIDO's memory should be turned to false? The problem is to get the right ones without unduly affecting teaching relating to other situations. We decided that all the elements corresponding to features which have occurred and which indicate the wrong answer should be turned false. This may appear to be a drastic solution, but it is guaranteed to turn off the right booleans. Since FIDO is quick to learn, these values will rapidly be turned true again if this is necessary.

g. Bitmap FIDO Gives an Easy Second Choice

One advantage of using a method which scores each message is that if the teacher of FIDO asks for a second choice this can be easily provided by giving the answer with the second highest score.

h. Bitmap FIDO is a Linear Decision Function

If we thought of the output of features extracted as a vector instead of a set, then we can see that Bitmap FIDO is a linear decision function as described in section 3.4. The vector $X = (X_1, X_2, X_3 \ldots X_n)$ would have an element to represent each feature.

X_1 would be 1 if F_1 had occurred, otherwise X_1 would be zero. For example, suppose F_1 and F_3 occurred, X would be (1, 0, 1, 0, 0). There would be a weight vector W^j for each message M_j. M_k would be selected if $X.W^k > X.W^m$ for all $m \neq k$. The memory update algorithm would place appropriate fractions into the W vectors. Fig. 5.7. shows a Bitmap FIDO's memory and the corresponding weight vectors for a FIDO's memory using a linear decision function.

	M_1	M_2	M_3	M_4
F_1	T	T	F	F
F_2	F	F	T	T
F_3	T	F	F	F
F_4	F	F	T	F

Bitmap FIDO's Memory

$W^1 = (½, 0, 0, ½, 0)$
$W^2 = (½, 0, ½, 0, 0)$
$W^3 = (0, ⅓, ⅓, 0, 0)$
$W^4 = (0, 1, 0, 0, 0)$

Corresponding weight vectors

Fig. 5.7 Bitmap FIDO's memory and weight vectors for a linear decision function.

5.4 Practical Considerations When Implementing a Bitmap FIDO

When implementing a Bitmap FIDO the following points should be considered:

a. The Nature of Features

The algorithms given so far number the features between one and some maximum. In practice it would be annoying for the programmer of the input mehcanism to have to place all his features on one scale. As will be seen when LR(1) syntax error messages are discussed, features can be divided up into several categories. That is features will be extracted by different algorithms and the natures of what the features in different categories represent will be different. To make things easier for the programmer of the input mechanism, FIDO should be able to partition different categories of input in its memory. Each of these categories we shall call a feature class. They will be discussed later when we consider interfacing to FIDO and the organisation of FIDO's memory.

Other important considerations when implementing a FIDO are the total number of features and how many features will occur at the same time. Our experience with FIDO suggests that FIDO performs best if only a small proportion of the features occurs at any one time. If this is the case, it may be feasible to consider placing FIDO's memory on a backing store and thus reduce the amount of main memory required.

b. The Number of Messages

The number of messages must affect the nature of FIDO's memory. For handling syntax error messages for LR(1) parsers the number of messages required is relatively small. Our test system can handle 47 messages and this seems adequate for a subset of BASIC. On the other hand, a question answer system may require considerably more messages. We will consider alternative memory organisation for small and large numbers of messages in the next section.

c. Organisation of Bitmap FIDO's Memory

In our sample algorithms for Bitmap FIDO, FIDO's memory was represented by a boolean array. To represent FIDO's memory as a boolean array in most high level language systems would be inefficient. For example, in B6700 Extended Algol (Burroughs 1974), each boolean array element requires one 48 bit word. Since each element in FIDO's memory can have only two states, only one bit is required.

Our demonstration implementation of FIDO uses one B6700 word for each feature. As the words are 48 bits long up to 48 messages can be represented by bits in the word. If more messages were required, a double precision word could be used to represent up to 96 messages. Therefore, in the B6700, FIDO's memory could be represented as a one dimensioned array. When planning the organisation of FIDO's memory we must consider the number of features. If there are a thousand or even five thousand features, it would be possible to implement FIDO's memory as an array held in store. If however, there are a million features, such an array is totally impracticable. As discussed earlier, we expect that only a few features will occur at any one time. It is therefore not unreasonable to hold a large array on disk.

Our B6700 version of FIDO uses a two-dimensional array to represent FIDO's memory. One axis represents feature classes, and the other features in those classes. The Bitmap to messages is stored in individual bits within a word. Normally, a problem with this technique would be that different feature classes might contain different numbers of features and thus some storage is wasted. This was not a problem in B6700 Extended Algol because any array row (feature class) could be RESIZED to hold the correct number of features. An alternative method would be to have FIDO perform the necessary calculations to convert features from the class-and-feature form into a single number which could be used as an index into a one-dimensional array.

An alternative organisation of FIDO's memory would be for FIDO to store the numbers of the messages indicated against features. In this form each array row would represent a feature and each element of that row would contain a message number. For example if feature F_1 indicates messages 2, 4 and 5 then row 1 of the memory array would contain the numbers 2, 4 and 5. Fig. 5.8a shows FIDO's memory in Bitmap form and Fig. 5.8b shows FIDO's memory in the alternative form.

This system would be advantageous if most features indicate only a small number of messages. It might be that if a feature indicates more than a certain number of messages, that feature will have no significant effect on the scoring algorithm (i.e. if that feature were omitted the same message would be selected). If this assumption can be made then FIDO's memory could be stored in a two-dimensional array of fixed size.

d. Interfacing Bitmap FIDO to Other Systems

It is intended that, apart from the input mechanism (feature extraction), an implementation of FIDO should be usable for a number of applications. This requires some standard interface between FIDO and other modules in the computer system.

FIDO must be informed:

	M_1	M_2	M_3	M_4	M_5	M_6
F_1	T	T	T	T	T	T
F_2	T	F	F	F	F	F
F_3	T	F	T	F	F	T
F_4	F	T	F	F	F	T
F_5	F	F	F	F	T	F
F_6	F	F	T	F	F	T
F_7	F	F	F	F	F	F
F_8	F	F	F	T	F	T
F_9	F	F	F	F	F	F

(a) Bitmap form of memory

FEAT	MESSAGE					
F_1	1	2	3	4	5	6
F_2	1					
F_3	1	3	6			
F_4	2	6				
F_5	5					
F_6	3	6				
F_7						
F_8	6	4				
F_9						

(b) Alternative form of memory

Fig. 5.8 A comparison of two possible ways of representing FIDO's memory

— which features have occurred

— which files or devices to write messages to and receive feedback from

— whether this is an initialisation call

— whether FIDO is in training or operating mode.

FIDO may be required to return the number of the message which was the correct answer. It will be seen when we discuss question-answer systems that this information is required as subsequent input to FIDO.

The form of the interface to FIDO must depend on the computer system being used. Our main implementation of FIDO is written in BURROUGHS EXTENDED ALGOL and run on a B6700 computer using the CANDE (Burroughs, 1971) message control system to interact with the user. Some test systems have also been implemented on a Data General ECLIPSE S/130 system running RDOS (Data General, 1971) and written in Data General's implementation of ALGOL 60 (Data General, 1971).
In both cases the numbers of the features which have occurred were passed into FIDO in a one-dimensional array which we will refer to as FEATURES. FIDO assumes that the array FEATURES starts at element zero and continues until element N, where FEATURES [N + 1]contains zero. As mentioned earlier it is desirable that features be divided up into feature classes. We did this by the convention that if FEATURES (I) was negative then the absolute value of FEATURES (I) was the feature class for subsequent features in the array. For example if FEATURES contain:

−1, 1, 3, 5, −2, 8, 7, 1, −3, 1, −1, 4, 0

then features 1, 3, 4 and 5 of feature class 1, feature 8, 7 and 1 of class 2, and feature 1 of class 3 have occurred.

This system of passing features to FIDO is easy to specify and it is easy for the writer of the input module to put features in such an array. For example, feature extraction for LR(1) parser error messages took about 25 lines of Burroughs Extended Algol code.

e. The Form of the Dialogue

As pointed out in section 4.3 the form of the dialogue used is important. The user must be sure whether his inputs are going to FIDO or to some other part of the system. For this reason, FIDO's inputs should be prompted by a unique prompt. In our example we have used '*' as a prompt for FIDO and '+' as a prompt for the other part of the system. On being called, FIDO should output a message number and a message. It is important that the message number be given as sometimes it is required by the teacher — for example to change the wording of a message or for subsequent teaching. If FIDO was in operating mode it would immediately return control to the calling system. If on the other hand FIDO was in training mode the prompt should be given. The user must be able to respond in a way which is both clearly defined and easy to remember. The form we have used and found successful is represented by the grammar in Fig. 5.9.

```
<Feedback>::=<Yes feedback>!
                <No try again>!
                <No correct message>!
                <No new message>!
                <Change command>!
                <Remove command>!
                <Join command>!
                <List command>!
                <Blank line>

<Yes feedback>::= YES!Y
<No try again>::= NO!N
<No  correct message> ::= <Message number>
<No new message>::="<Message text>
<Change command>::= CHANGE <New line> <Message text>
<Remove command>::= REMOVE <Message number>
<Join command>::= JOIN <Message number> <Message number> <Newline> <Messagetext>
<List command>::=LIST!L
<Blank line>::=<End of line character>
<Message number>::=<Digit>!<Digit><Digit>
<Digit>::=1!2!3!4!5!6!7!8!9!0
<Message text>::=<Any characters up to the end of line>
<Newline>::=<End of line character>
```

Fig. 5.9 Grammar of feedback teacher can give

The meaning of these commands are:

1. < Yes feedback> , < No try again> , < No correct message> and < No new message> are described in section 4.2.
2. The < List command> will list the numbers and texts of messages FIDO has been taught.
3. The < Change command> will change the wording of the text of a message.
4. The < Remove command> will remove the message from FIDO's memory.
5. The < Join command> will mean that all the teaching associated with the second < Message number> will be transferred to the first. The text of this message will be changed to <Message text>.
6. <Blank line> will mean that FIDO's memory should not be updated.

CHAPTER VI

EXAMPLES OF FIDO IN USE

In this chapter we discuss a number of examples of the use of FIDO. The chief example is that of using FIDO to give error messages for LR(1) parsers. The language used is a subset of MINI-BASIC (Lewis 1976), with the grammar as given in Appendix A. The grammar has been chosen to be small enough that a complete set of messages can be taught in a reasonable time, yet it is complex enough to test FIDO. A second example is given, also using a LR(1) parser, but for a modified version of VORTEX, the job control language for the Varian mini computer (Varian, 1974).

In both these examples:

a. The lexical analyser maps individual characters onto the lexical item numbers used by the parser.
b. Blanks are ignored.
c. A character not recognised is mapped onto a special lexical item.
d. The end-of-line character is treated as the empty input string by the parser but is printed as a colon on the extreme right of the page.
e. It is assumed that an error is reported at the first character which cannot be matched by the parser.

We then discuss a simple question-answer system written to illustrate the versatility of FIDO. Finally we re-examine performance criteria in the light of how well FIDO meets the goals set out in Chapter 4.3.

6.1 Error Messages For An LR(1) Parser

This section discusses our chief example, learning error messages for an LR(1) parser. The language used is a subset of MINI-BASIC. Its grammar is given in Appendix A and the result of the teaching session in Appendix B.

The features passed to FIDO from the parser's error routines are:

a. The Look Ahead Character

The features in this feature class represent the last lexical item looked at by the parser. Referring to the description of LR(1) parser in Section 2.2, this feature corresponds to the left most character in w. It was the input of this item which resulted in the break-down of the parse. Since each lexical item is represented by an integer, the feature extraction algorithm simply uses that integer value as the feature number. It is important that this item be represented as a feature as this may be an indication of what the user thought he was doing as opposed to what he is allowed to do. For example, if the user types:

```
10 LET A = B =
```

an appropriate error message would be

MULTIPLE ASSIGNMENTS ARE NOT ALLOWED

On the other hand if the user typed

```
10 LET A = (
```

an appropriate error message would be

INVALID OPERATOR

b. The Table Controlling the Stack

The features in this feature class represent the table which was controlling the stack when the parse breaks down (i.e. the look ahead character was looked up in this table and an error indicator was found). In terms of our formal description of the LR(1) parser this is the rightmost element in s. The table numbers are represented by integers which are used as feature numbers. This table indicates what productions in the grammar were being matched when the parse broke down. The importance of this feature class is discussed further in the next section.

c. Tables on the Stack

The features in this class represent all the tables on the stack except for the table controlling the stack. Feature extraction for this class proceeds by scanning the stack and representing features by the numbers on the stack. Each table in an LR(k) parser corresponds to the parser's matching up to a particular point in some production of the grammar. For example, Table T_a might represent that the parser has matched the production

<STATEMENT> : : = LET<VARIABLE> = <ARITHMETIC EXPRESSION>

up to where the arrow indicates. It might be that the string being parsed will match two productions. For example Table T_b might represent

<VARIABLE> : : = <LETTER>

<VARIABLE> : : =<LETTER><DIGIT>

If the parser had just parsed the string

LET A

the stack would contain T_a and T_b as well as other tables representing productions which were being matched. If the next character in the string was invalid (i.e. could could not be matched) the parse would break down. Therefore when the parser detects an error the tables on the stack pinpoint the places in the grammar which were being matched when the error occurred. Often the most important table on the stack is the one on the top. Freeth's system discussed in Chapter 2, uses only the table on the top of the stack to determine the error message (Freeth's system sometimes does reductions in error situations thus revealing other tables on the top of the stack. Any error messages attached to these tables are also output.) We feel that this table is important enough to be represented by a feature class of its own. The tables further down the stack give more indication of the type of error and are all represented in one feature class.

As an example of where these features are important, suppose the user types "LET AB", a suitable error message might be

VARIABLE NAME IS LETTER OR LETTER DIGIT

Table T_b on the top of the stack would be important in deciding on this error message. If however, the user typed

LET A+

and the error message was

THE FORM OF A LET STATEMENT IS LET VARIABLE =
ARITHMETIC EXPRESSION

it would be the table T_a further down the stack which would be most important.

d. Possible Next Lexical Items

The features in this feature class represent lexical items which, if typed instead of the one error, would be accepted by the parser. Feature extraction finds these features by looking up the parse action (f) part of the table controlling the stack. Any lexical item which does not contain an error indication when looked up in this table is a possible valid next item. This feature class was added to FIDO to help FIDO generalise its messages. For example, suppose a user types

LET

A possible next item would be any of the letters. A suitable error message might be

VARIABLE REQUIRED NEXT

Suppose the user now types

FOR

Again possible next items would be any of the letters and these features would indicate the above message.

One possible disadvantage of having this feature class in the system is that often a large number of features occur at once thus slowing down FIDO's response time.

6.2 Discussion

Two hours of teaching FIDO error messages for our subset of mini BASIC is given in Appendix B. This section discusses these results and our experiences in teaching FIDO.

a. Use of Change and Join

While the inclusion of change and join commands are not part of the main concept of FIDO, they are important to provide a flexible and easy-to-use system. Fig. 6.1 illustrates the use of these commands. In both cases it is still necessary to tell FIDO the correct answer after using the commands. The change command is generally used in two situations.

The first is when the teacher wishes to correct a simple mistake he has made (e.g. a spelling mistake). The second is when the user wants to change the meaning of a message. For example Fig. 6.1 (a) shows the meaning of message 4 being generalised from being about GOTO only to being about both GOTO and GOSUB.

The JOIN command is typically used if two messages of very similar meaning are taught by the teacher. The teacher may decide that it is more convenient to have both these as the same message (e.g. Fig. 6.1 (b)).

b. Use of 'No'

While teaching FIDO we found the 'No. Try again' feedback very useful. This feedback to FIDO allows the teacher to look at a number of outputs from FIDO in decreasing order of probability. Because Bitmap FIDO's memory is not updated until FIDO is told the correct answer, a teacher can say that one of the messages for which he previously entered 'No' is, in fact, the correct answer. The use of the

```
*30 GO TO T
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*"GO TO SHOULD BE FOLLOWED BY STATEMENT NUMBER


*G
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


*50 GOSUB T
 3:"GO TO SHOULD BE FOLLOWED BY STATEMENT NUMBER
*CHANGE 3
"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
*3


*60 GOSUB T
 3:"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
*Y
```

(a) Use of CHANGE command.

```
*99 FOR I = 1.4.
16:"REPEATED DECIMAL POINT
*Y


*100 LET A=B+12.3E4.
17:"UNEXPECTED DECMIAL POINT
*L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
 3:"GO TO OR GOSUB SHOULD BE FOLLOWED BY A STATEMENT NUMBER
 4:"VARIBLE NAMES ARE LETTER OR LETTER DIGIT
 5:"REMARKS MUST CONTAIN LETTERS ONLY
 6:"MISMATCHED PARENTHESIS
 7:"INVALID OPERATOR
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
 9:"MISSING OPERATOR OR PARENTHESIS
10:"TO EXPECTED
11:"STEP EXPECTED
12:"LET STATMENT SHOULD BE <VARIABLE>=<EXPRESSION>
13:"FOR SHOULD BE FOLLOWED BY A <VARIABLE>
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
15:"MISSING OPERATOR
16:"REPEATED DECIMAL POINT
17:"UNEXPECTED DECMIAL POINT
*JOIN 16 17
"UNEXPECTED DECMIAL POINT
*16


 *99 LET A=1.2.
 16:"UNEXPECTED DECMIAL POINT
 *Y
```

(b) Use of JOIN command.

Fig. 6.1 The use of the JOIN command and CHANGE command.

'No' command saves the teacher having to use the more time consuming LIST command.

c. Difficult to Decide on Messages

When the reader looks at the examples, he may feel that we have not taught FIDO the best possible messages. Part of our aim for FIDO is that it be flexible enough to cater for this situation. If the reader regularly used this parser, he might decide to retrain FIDO to give what he feels is a better set of error messages. After regular use of the parser we may ourselves feel inclined to change the messages. Before FIDO was developed, we assumed that it would be easy to decide the messages with which FIDO should be trained. However, while training FIDO, we found it very difficult to decide what message to give. For example, suppose that we type

10 FOR I =3S

Possible messages may be

'TO' EXPECTED

MISSING OPERATOR

VARIABLE SHOULD BE<LETTER>|<LETTER><DIGIT>

Alternatively we might decide on a more general message such as

THE FORM OF A FOR STATEMENT IS FOR< VARIABLE> =< ARITHMETIC EXP> TO <ARITHMETIC EXP>

The difficulty in deciding on error messages arose in spite of the fact that we had the error situation in front of us. Using a normal error message system the designer must guess at the exact situation before deciding on error messages. We feel that this additional level of difficulty adds to the problem of poor error messages. Frequently, the designer appears not to try to see how the user of the system will see the error, but gives error messages in terms of the operation of the parser. Fig. 6.2 from the B6700 BASIC Compiler is a good example. If the user did not know what an arithmetic primary was he would have difficulty in deciding what the error messages mean. We have tried to teach FIDO a fairly detailed set of error messages and have probably given them in more detail than most BASIC interpreters and compilers. For example, Data General's BASIC interpreter appears to have only two error messages.

d. Apparently Unreasonable Messages from FIDO

In some situations it is impossible for FIDO to give a reasonable and meaningful message because one has not been taught, but FIDO will still output the message with the highest score (e.g. Fig. 6.3(a)). Thus FIDO must sometimes give incorrect messages. Of more concern are the situations in which FIDO does have a reasonable message to give, but gives the wrong message (Fig. 6.3(b)). While the message given is unsatisfactory, a careful study of previous teaching may show the message was not completely unwarranted.

Such messages also appear in conventional error message giving systems. For example, refer back to Fig. 1.1. Referring to Fig. 6.3(b), suppose that instead of

MULTIPLE ASSIGNMENTS NOT ALLOWED

We had taught FIDO

THIS SYMBOL IS NOT ALLOWED IN ARITHMETIC EXPRESSIONS

Then the message would be a more reasonable one.

The big advantage of our system is that FIDO can always be trained a little further until a more reasonable message is given.

```
      10 LET A = A+

-------------------------->
**********    ARITHMETIC PRIMARY EXPECTED IN EXPRESSION
      20 LET A = A + BC
-------------------------->
**********    END OF STATEMENT WAS EXPECTED
      30 LET A = (A+ B
-------------------------->
**********    RIGHT PARENTHESIS MISSING FOR ARITHMETIC PRIMARY
      40 LET A= B = C
      50 LET = B+-
------------------------->
**********    ILLEGAL TARGET VARIABLE NAME IN LET STATEMENT
      60 LET A = (B+-C))
-------------------------->
**********    ARITHMETIC PRIMARY EXPECTED IN EXPRESSION
--------------------------->
**********    RIGHT PARENTHESIS MISSING FOR ARITHMETIC PRIMARY
      70 LET A=(B+(*D))
-------------------------->
**********    ARITHMETIC PRIMARY EXPECTED IN EXPRESSION
--------------------------->
**********    RIGHT PARENTHESIS MISSING FOR ARITHMETIC PRIMARY
--------------------------->
**********    RIGHT PARENTHESIS MISSING FOR ARITHMETIC PRIMARY
      80 END
********************  BASIC COMPILATION SUMMARY  ********************
```

Fig. 6.2 Example of error messages in terms of the parser.

e. Computer Resources

Appendix B comprises the results of two hours of teaching. In this time FIDO used 22.8 seconds of processor time and was called more than 66 times so that each call used less than 0.3 seconds. All of these calls were in training mode. We feel that this is not an unreasonable demand on computer resources.

6.3 Job Control Language For Vortex

The job control language for VORTEX was chosen for our second example because it is typical of small scale command languages. Its grammar is given in Appendix C and the result of the teaching session in Appendix D. While teaching these error messages, we tried to express them in such a way that they would tell the user how to correct the mistake he has made. Appendix D also has a section illustrating FIDO in operating mode.

6.4 FIDO for a Question/Answer System

A program which we refer to as TALK was written to enable FIDO to function as a

question/answer system. That is, the user can type a question and the system will respond with an answer to that question. Appendix E gives a sample interaction with this system. The algorithm of TALK is very simple. Firstly, TALK reads a line from a terminal. Next it delimits words in that line by blanks. Each word is looked up in a dictionary to find a number representing that word. If the word is not found in the dictionary, it is placed into it. For example, if the user typed

HOW ARE YOU TODAY

this line might be converted into numbers

3, 5, 2, 8

These numbers would be used as feature class 1.

Next, these numbers are paired as:

(3,5), (5,2), (2,8)

Each pair is given a unique number. These numbers are used as features in class 2.

Feature class 3 represents triplets:

(3, 5, 2), (5, 2, 8)

and so on up to feature class 5.

In addition, TALK creates two other feature classes. One represents the last message given and the other the last three messages given by FIDO. When TALK has created the array FEATURES it calls FIDO.

This scheme is too simple to provide a really successful question/answer system, but the result has been a system which can perform remarkably well with very little development. Appendix E is a result of a teaching session with TALK.

6.5 Meeting of Performance Criteria

The most important question when assessing a system such as FIDO is to decide how well it met its design goals. The best way to answer this question is to compare actual performance with the performance criteria. Our criteria are laid down in section 4.3.

a. Easy to Learn to Use

All the examples given in the Appendices were done by persons other than the author. These people required only a few minutes instruction – so we conclude that this goal has been met.

b. Natural to Use

While it could be argued that the way that one responds to FIDO is not the same as that in which one would respond to a person, people using FIDO have found that the responses required are sufficiently natural to state that FIDO is easy to use. One slight exception to this is the 'Yes' input. Teachers of FIDO would occasionally forget to respond to correct answers. It is therefore important that FIDO checks any responses from the teacher and informs him if it is invalid.

c. Good Response Time

Response time when training FIDO was usually adequate. The major problem here was due to variations caused by the loading of the B6700 on which FIDO was run. We

found the response time entirely satisfactory when the B6700 was under light load.

d. Learns Well

This criterion is rather subjective and it is therefore hard to say whether or not it has been met. The feeling of those who have taught the system is that it learns adequately but could be improved. The reader must draw his own conclusions after studying the examples given in the Appendices.

e. Economic on Computer Resources

FIDO performed adequately in a time-sharing environment. As already discussed, processor demands are reasonable.

CHAPTER VII

CONCLUSION

In the preceding chapters we have discussed a common problem: that messages output from a computer are frequently of poor quality. We then went on to propose that this problem be solved by teaching a computer system good messages. A brief survey of adaptive and learning mechanisms was presented, and some problems with these ideas identified. After examining the requirements of a learning system such as FIDO we went on to propose a practical method of implementing a FIDO. The evidence that FIDO is a practical system can be found by examining the examples given.

The FIDO proposed in this thesis is rather simple, but on the other hand it makes very reasonable demands on computer resources. One observation the author made while examining adaptive systems is that most systems reported in the literature are very theoretical or require much computing power. When designing FIDO we tried to overcome this problem. FIDO will function economically provided that the number of messages is not too large. One possible direction for further research would be to try to overcome this limitation. The other limitation of FIDO lies in the scoring algorithm. This algorithm has no method to assess the importance of a feature which has not occurred. It is important that any solutions to this problem should not significantly increase processing time.

Count FIDO, described in Chapter 5, appears to us to be a system with very little potential. However, Tree FIDO appears to have a powerful memory structure if some satisfactory method for memory update could be found. Tree FIDO might well prove to be the foundation for a powerful FIDO system.

We will conclude this thesis with a brief comment about one direction of computer development and how we see FIDO fitting into this. With the availability of cheap micro-processors it is now economically feasible to have a computer in every home (much like radio, television and telephone). So far, personal computing has only been taken up by a small number of hobbyists. One reason for this is that to use any computer system a person must learn special programming languages. The author feels that the personal computer will only become a reality when it is adapted to the requirements of people. Since different people have different requirements and an individual's requirements change over time, it would be desirable that the computer adapts itself to the requirements of the user. This thesis contributes to this by showing that simple and economic adaptive systems can work. While FIDO only solves a small part of the man-computer interaction problem, the author hopes that it may act as a stepping stone to a more complete system.

REFERENCES

Anderberg M.R. (1973). *Cluster Analysis for Applications.* Academic Press, 1973.

Aho A.V. and Ullman J.D. (1972). *The Theory of Parsing, Translation and Compiling, Volume 1: Parsing.* Prentice-Hall, 1972.

Andreae J.H. and Cleary J.G. (1976). "A New Mechanism for a Brain". *International Journal of Man-Machine Studies,* vol. 8 No. 1, January 1976, pp 89-119.

Arkadev A.G. and Braverman E.M. (1967). *Teaching Computers to Recognize Patterns.* Translated from Russian by Turski and Cowan. Academic Press, 1967.

Biermann A.W. (1972a). "On the inference of Turing Machines". *Artificial Intelligence Vol.3* No. 3 pp 181-198, Fall 1972.

Biermann A.E. and Feldman J.A. (1972b). "On the Synthesis of Finite-State Machines from Samples of their Behaviour". *IEEE Trans. on Computers,* Vol C21 No. 6, June 1972. pp 592-597.

Biermann A.W. and Feldman J.A. (1972c). "A Survey of Results in Grammatical Inference". *Frontiers of Pattern Recognition,* Academic Press 1972, pp 31-54. Watanabe S. editor.

Burroughs (1971). *B6700/7700 Command And Edit (CANDE) Language Information Manual.* Burroughs, 1971.

Burroughs (1974). *Burroughs B6700/7700 Algol Language Reference Manual.* Burroughs, 1974.

Cover T.M. and Hart P.E. (1967). "Nearest Neighbour Pattern Classification". *IEEE Trans. on Information Theory,* Vol. IT13 No 1, January 1967.

Data General (1971). *ECLIPSE Line Extended Algol Users Manual.* Data General, 1972.

DEC (1976). *DEC System-20 User Guide DEC-20-OUGAA-A-D.* Digital Equipment Corporation.

Eland D.R. (1975). *An Information and Advising System for an Automated Introductory Computer Science Course.* Thesis, PhD., University of Illinois, 1975.

Freeth G.D. (1978) .*Error Handling for LR(k) Parsers used in Conversational/Interactive Systems.* Thesis MSc, University of Canterbury.

Fu K.S. and Booth T.L. (1975). "Grammatical Inference: Introduction and Survey Parts I and II" *IEEE Trans. on Systems, Man and Cybernetics.* vol SMC5 No. 1, January 1975, pp 95-111 and Vol SMC5 No. 4, July 1975, pp 409-423.

Fu K.S. and Rosenfeld A. (1976). "Pattern Recognition and Image Processing". *IEEE Trans. on Computers,* Vol C25 No. 12, December 1976, pp 1336-1346.

Ho Y.C. and Agrawala A.K. (1967). "On Pattern Classification: Introduction and Survey". *Proc. IEEE* vol 26 No. 12, December 1968, pp 2101–2114.

Johnson V.J. (1969). *Computer Graphics in Logic Circuit Design.* Thesis, MSc. University of Alberta, 1969.

Lewis P.M., Rosenkrantz D.J., and Stearns R.E. (1976).*Compiler Design Theory.* Addison-Wesley Publishing Company, 1976.

Lindsay R.K. (1973). "In Defence of Ad Hoc Systems". Shank R.C. and Colby K.M. editors, *Computer Models of Thought and Language.* W.H. Freeman and Company 1973.

Martin J. (1973). *The Design of Man-Computer Dialogues.* Prentice-Hall 1973.

Poonen G. (1977) "Error Recovery for LR(k) Parsers". Gilchrist B. editor, *Information Processing 77,* North-Holland Publishing Company 1977, pp 529-533.

Raphael R. (1965). "SIR: Semantic Information Retrieval". Minsky M., Editor. *Semantic Information Processing.* The MIT Press 1968.

Salton G. editor (1971). *The Smart Retrieval System : Experience in Document Processing.* Prentice Hill, 1971.

'Slagle J.R. (1971). *Artificial Intelligence: The Heuristic Programming Approach.* McGraw-Hill Book Company, 1971.

Spencer D.D. (1968). *The Computer Programmers Dictionary and Handbook.* Blaisdell Publishing Company 1968.

Starkweather J.A. (1969). "A Common Language for a Variety of Conversational Needs". *Computer Assisted Instruction:* A book of Readings. ed. Atkinson & Wilson. Academic Press 1969.

Tou J.T., and Rafael C.G. (1974). *Pattern Recognition Principles.* Addison-Wesley Publishing Company, 1974.

Uhr L., (1973). *Pattern Recognition, Learning and Thought.* Prentice-Hall 1973.

Varian (1974). *VORTEX Reference Manual.* Manual Data Machines. 1974.

Weizenbaun J. (1966). "ELIZA – A computer Program for the Study of Natural Language Communications Between Man and Machine", *Comm. ACM.* Vol. 9 No. 1., January 1966 pp. 36-45.

Yu C.T. and Salton G. (1977). "Effective Information Retrieval Using Term Accuracy". *Comm. ACM.* Vol 20 No. 3, March 1977, pp 135-142.

APPENDIX A

GRAMMAR FOR SUBSET OF MINI BASIC

(Used for the example given in Appendix B)

This language was chosen to provide a comprehensive test of FIDO's ability to learn syntax error messages for systems using LR(1) parsers. The language is a subset of MINI BASIC (Lewis, 1976). The meanings of statements are the same as for equivalent BASIC statements. As our interest in this thesis is confined to syntax errors, only the parser has been implemented. The grammar is:

```
<START>::=<INP>
<INP>::=<NUMBER><STATEMENT>
<STATEMENT>::=
<STATEMENT>::=LET<VARIABLE>=<EXPRESSION>
<STATEMENT>::=GOTO<NUMBER>
<STATEMENT>::=IF<EXPRESSION><RELATIONAL OPERATOR><EXPRESSION>
  GOTO<NUMBER>
<STATEMENT>::=GOSUB<NUMBER>
<STATEMENT>::=RETURN
<STATEMENT>::=FOR <VARIABLE>=<EXPRESSION>TO<EXPRESSION>
<STATEMENT>::=FOR<VARIABLE>=<EXPRESSION>TO<EXPRESSION>STEP<EXPRESSION>
<STATEMENT>::=NEXT<VARIABLE>
<STATEMENT>::=REM<CHARACTERS>
<EXPRESSION>::=+<TERM>
<EXPRESSION>::=-<TERM>
<EXPRESSION>::=<EXPRESSION>+<TERM>
<EXPRESSION>::=<EXPRESSION>-<TERM>
<EXPRESSION>::=<TERM>
<TERM>::=<TERM>*<FACTOR>
<TERM>::=<TERM>/<FACTOR>
<TERM>::=<FACTOR>
 <FACTOR>::=<FACTOR>@<PRIMARY>
<PRIMARY>::=(<EXPRESSION>)
<PRIMARY>::=<VARIABLE>
<PRIMARY>::=<UNSIGNED CONSTANT>
<NUMBER>::=<NUMBER><DIGIT>
<NUMBER>::=<DIGIT>
<UNSIGNED CONSTANT>::=<NUMBER><EXPONENT>
<UNSIGNED CONSTANT>::=<NUMBER>.<NUMBER><EXPONENT>
<UNSIGNED CONSTANT>::=.<NUMBER><EXPONENT>
<EXPONENT>::=E+<NUMBER>
<EXPONENT>::=E-<NUMBER>
<EXPONENT>::=E<NUMBER>
<EXPONENT>::=
<VARIABLE>::=<LETTER>
<VARIABLE>::=<LETTER><DIGIT>
<RELATIONAL OPERATOR>::==
<RELATIONAL OPERATOR>::= ^=
<RELATIONAL OPERATOR>::= <
<RELATIONAL OPERATOR>::= <=
<RELATIONAL OPERATOR>::= >
<RELATIONAL OPERATOR>::= >=
<DIJIT>::=1|2|3|4|5|6|7|8|9|0
<LETTERS>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<CHARACTERS>::=<CHARACTERS><LETTER>
<CHARACTERS>::=<LETTER>
```

APPENDIX B

RESULT OF TEACHING ERROR MESSAGES FOR MINI BASIC

This Appendix illustrates the teaching of syntax error messages to FIDO for the subset of MINI BASIC given in Appendix A. The input to the LR(1) parser is prompted by a "+" which appears as the first character of a line. The input to FIDO is prompted by "*". The parser types "ALL RIGHT" if the string entered to the parser is syntactically correct. Apart from these and the second line of the "JOIN" and "CHANGE" commands, all other lines are output by FIDO.

We stress that this session, and those documented in subsequent appendices are the work of people other than the author. We feel that this is a very fair test, especially since people, under these conditions, seem to be keen to try and "beat" the system as they train it.

```
+L
NO MORE MESSAGES
*"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER


+L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


 10 LET A = B
ALL RIGHT


+A
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+20 A
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*"NO VALID STATEMENT CAN BEGIN WITH THIS


+30 GO TO T
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*"GO TO SHOULD BE FOLLOWED BY STATEMENT NUMBER


+G
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+50 GOSUB T
 3:"GO TO SHOULD BE FOLLOWED BY STATEMENT NUMBER
*CHANGE 3
"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
*3


+60 GOSUB T
 3:"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
*Y


+69 IF GO
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*"VARIABLE NAMES ARE LETTER OR LETTER DIGIT


+70 LET TE
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*Y
```

```
+80 FOR TE
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*Y


+99 REM IT IS 3
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*"REMARKS MUST CONTAIN LETTERS ONLY


+88 IF (A=(B+3) =
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*"MISMATCHED PARENTHESIS


+99 LET A = ((B+2)/(C=6)))
 6:"MISMATCHED PARENTHESIS
*Y


+99 LET A = ((B+2)/(C=
 6:"MISMATCHED PARENTHESIS
*LIST
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
 3:"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
 5:"REMARKS MUST CONTAIN LETTERS ONLY
 6:"MISMATCHED PARENTHESIS
*"INVALID OPERATOR


+99 IF A = B =
 6:"MISMATCHED PARENTHESIS
*NO
 7:"INVALID OPERATOR
*Y


+99 LET A = B =
 6:"MISMATCHED PARENTHESIS
*"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED


+99 IF A = B =
 7:"INVALID OPERATOR
*Y


+L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+99 LET A = B =
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
*Y


+99 LET A = (B =
 6:"MISMATCHED PARENTHESIS
*N
 7:"INVALID OPERATOR
*Y


+99 LET A = (B =
 7:"INVALID OPERATOR
*Y


+99 IF A = (B =
 7:"INVALID OPERATOR
*Y


+99 LET A = B =
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
*Y
```

```
+99 IF A = B =
 7:"INVALID OPERATOR
*Y


+99 FOR I = J =
 7:"INVALID OPERATOR
*Y


+F
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+99 FOR I = (J = 2)*(K + 3 S
 7:"INVALID OPERATOR
*"MISSING OPERATOR OR PARENTHESIS


+99 FOR I = (J-2)*(K+3) S
 9:"MISSING OPERATOR OR PARENTHESIS
*"TO EXPECTED


+99 FOR I = J S
 7:"INVALID OPERATOR
*N
10:"TO EXPECTED
*Y


+F
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+99 FOR I = J S
10:"TO EXPECTED
*Y


+99 FOR I = J TO 9 D
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*N
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*N
 5:"REMARKS MUST CONTAIN LETTERS ONLY
*"STEP EXPECTED


+99 FOR I = 1 TO 10 STO
 7:"INVALID OPERATOR
*N
 9:"MISSING OPERATOR OR PARENTHESIS
*N
10:"TO EXPECTED
*N
 6:"MISMATCHED PARENTHESIS
*N
11:"STEP EXPECTED
*Y


+99 FOR I = 1 TO 10 STEO
11:"STEP EXPECTED
*Y


+99 FOR I = 1 TO 10*ST
11:"STEP EXPECTED
*N
10:"TO EXPECTED
*N
```

```
 7:"INVALID OPERATOR
*N
 9:"MISSING OPERATOR OR PARENTHESIS
*N
 6:"MISMATCHED PARENTHESIS
*N
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
*N
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*Y


+99 FOR I = 1 TO
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*N
 7:"INVALID OPERATOR
*N
11:"STEP EXPECTED
*NO
 9:"MISSING OPERATOR OR PARENTHESIS
*N
10:"TO EXPECTED
*Y


+99 LET X +
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*"LET STATEMENTS SHOULD BE <VARIABLE>=<EXPRESSION>


+99 FOR I+
12:"LET STATEMENTS SHOULD BE <VARIABLE>=<EXPRESSION>
*LIST
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
 3:"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
 5:"REMARKS MUST CONTAIN LETTERS ONLY
 6:"MISMATCHED PARENTHESIS
 7:"INVALID OPERATOR
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
 9:"MISSING OPERATOR OR PARENTHESIS
10:"TO EXPECTED
11:"STEP EXPECTED
12:"LET STATEMENTS SHOULD BE <VARIABLE>=<EXPRESSION>
*"FOR SOULD BE FOLLOWED BY <VARIABLE>


+99 LET I*
12:"LET STATEMENTS SHOULD BE <VARIABLE>=<EXPRESSION>
*Y


+99 FOR I/
13:"FOR SOULD BE FOLLOWED BY <VARIABLE>
*CHANGE 13
"FOR SHOULD BE FOLLOWED BY <VARIABLE>
*13


+10 LET A = B*/
 5:"REMARKS MUST CONTAIN LETTERS ONLY
*N
11:"STEP EXPECTED
*N
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*N
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*N
 7:"INVALID OPERATOR
*"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER


+10 LET A = B**
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*Y


 10 LET A = B*C
ALL RIGHT
```

```
+10 LET A = B*(*
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*Y


+10 IF A = B+=
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*Y


+10 GO TO 3+
 3:"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
*Y


+10 IF A==
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
 3:"GOTO OR GOSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
 5:"REMARKS MUST CONTAIN LETTERS ONLY
 6:"MISMATCHED PARENTHESIS
 7:"INVALID OPERATOR
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
 9:"MISSING OPERATOR OR PARENTHESIS
10:"TO EXPECTED
11:"STEP EXPECTED
12:"LET STATEMENTS SHOULD BE <VARIABLE>=<EXPRESSION>
13:"FOR SHOULD BE FOLLOWED BY <VARIABLE>
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*Y


+I
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


 10 IF A==B GO TO 3
ALL RIGHT


+10 IF A = 12B
11:"STEP EXPECTED
*N
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*N
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*N
 4:"VARIABLE NAMES ARE LETTER OR LETTER DIGIT
*"MISSING OPERATOR


+10 LET A = 4B
15:"MISSING OPERATOR
*Y


+10 FOR I = 1 TO 4J
11:"STEP EXPECTED
*Y


+10 IF A = 4B
15:"MISSING OPERATOR
*Y


+10 LET A = 3.1.
15:"MISSING OPERATOR
*N
11:"STEP EXPECTED
*"REPEATED DECIMAL POINT
```

```
+10 GU TO 3.
 3:"GUTO UR GUSUB SHOULD BE FOLLOWED BY STATEMENT NUMBER
*Y


+99 LET A=B.
 7:"INVALID OPERATOR
*Y


+00 LET A=9E3.
16:"REPEATED DECIMAL POINT
*"UNEXPECTED DECMIAL POINT


+99 IF A = 3.1.
16:"REPEATED DECIMAL POINT
*Y


+99 IF A = 3E1.
17:"UNEXPECTED DECMIAL POINT
*Y


+99 FUR I = 1.4.
16:"REPEATED DECIMAL POINT
*Y


+99 FUR A=1E.
16:"REPEATED DECIMAL POINT
*N
 7:"INVALID OPERATOR
*N
 4:"VARIBLE NAMES ARE LETTER OR LETTER DIGIT
*N
11:"STEP EXPECTED
*N
17:"UNEXPECTED DECMIAL POINT
*Y


+99 FUR A=1E3.
17:"UNEXPECTED DECMIAL POINT
*Y


+L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+100 LET A=B+12.3E4.
17:"UNEXPECTED DECMIAL POINT
*L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
 3:"GU TO OR GUSUB SHOULD BE FOLLOWED BY A STATEMENT NUMBER
 4:"VARIBLE NAMES ARE LETTER OR LETTER DIGIT
 5:"REMARKS MUST CONTAIN LETTERS ONLY
 6:"MISMATCHED PARENTHESIS
 7:"INVALID OPERATOR
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
 9:"MISSING OPERATOR OR PARENTHESIS
10:"TO EXPECTED
11:"STEP EXPECTED
12:"LET STATMENT SHOULD BE <VARIABLE>=<EXPRESSION>
13:"FOR SHOULD BE FOLLOWED BY A <VARIABLE>
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
15:"MISSING OPERATOR
16:"RLPEATED DECIMAL POINT
17:"UNEXPECTED DECMIAL POINT
*JOIN 16.17
"UNEXPECTED DECMIAL POINT
*16
```

```
+99 LET A=1.2.
16:"UNEXPECTED DECMIAL POINT
*Y


+99 LET A = B <
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
*N
 6:"MISMATCHED PARENTHESIS
*N
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*N
 7:"INVALID OPERATOR
*Y


+60 LET A=B> C
 7:"INVALID OPERATOR
*Y


+99 LET A=B+(C*"
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*Y


+99 LET A= B+)
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*N
 7:"INVALID OPERATOR
*N
 5:"REMARKS MUST CONTAIN LETTERS ONLY
*N
15:"MISSING OPERATOR
*N
11:"STEP EXPECTED
*N
16:"UNEXPECTED DECMIAL POINT
*N
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*N
 4:"VARIBLE NAMES ARE LETTER OR LETTER DIGIT
*N
 6:"MISMATCHED PARENTHESIS
*Y


+99 IF A)
 6:"MISMATCHED PARENTHESIS
*Y


+99 IF (3+2 =
 7:"INVALID OPERATOR
*Y


+99 IF (3+2 =
 7:"INVALID OPERATOR
*N
 6:"MISMATCHED PARENTHESIS
*Y


+99 IF (3+2 =
 6:"MISMATCHED PARENTHESIS
*Y


+44 IF 6+(2*B-0 =
14:"NO OPERATOR MAY IMMEDIATELY FOLLOW ANOTHER
*N
 7:"INVALID OPERATOR
*N
 6:"MISMATCHED PARENTHESIS
*Y
```

```
+99 IF (2+3 =
 6:"MISMATCHED PARENTHESIS
*Y


+30 IF A+(B@3@=
 6:"MISMATCHED PARENTHESIS
*Y


+10 S
 2:"NO VALID STATEMENT CAN BEGIN WITH THIS
*Y


+20 GO TO T
 3:"GO TO OR GOSUB SHOULD BE FOLLOWED BY A STATEMENT NUMBER
*Y


+30 LET TE
 4:"VARIBLE NAMES ARE LETTER OR LETTER DIGIT
*Y


+40 REM IT IS 3
 5:"REMARKS MUST CONTAIN LETTERS ONLY
*Y


+60 LET A.
 7:"INVALID OPERATOR
*Y


+70 LET C=D=
 7:"INVALID OPERATOR
*N
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
*Y


+L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+10 LET X=Y=
 7:"INVALID OPERATOR
*N
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
*Y


+L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+10 LET G=Q=
 8:"MULTIPLE ASSIGNMENTS ARE NOT ALLOWED
*Y


+L
 1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+10 LET A=B.
 7:"INVALID OPERATOR
*Y
```

```
+50 FOR I=1S
10:"TO EXPECTED
*Y


+10 LET A=3.1.
16:"UNEXPECTED DECMIAL POINT
*Y


+L
  1:"ALL STATEMENTS MUST BEGIN WITH A STATEMENT NUMBER
*Y


+10 LET A=4B
15:"MISSING OPERATOR
*Y
```

APPENDIX C

GRAMMAR FOR JOB CONTROL LANGUAGE FOR VORTEX

(Used for example given in Appendix D)

This language is based on the VORTEX job control language and was chosen for our second example because we feel it is typical of many small scale command languages. The language is used to direct the operating system to perform various functions.

For example:

/JOB,MY JOB signals the start of a new job called "MY JOB."

/MEM,3 enlarges the memory allocated to the next program by 3 X 512 words.

/ASSIGN,3,FRED allows the disk file FRED to be referenced as logical unit 3.

/SREC,3,4,F causes four records to be skipped on logical unit 3.

The grammar for the language is:

```
<INPUT>::=/<COMMAND>
<COMMAND>::=JOB,<NAME>
<COMMAND>::=ENDJOB
<COMMAND>::=C,<COMMENT>
<COMMAND>::=MEM,<INTEGER>
<COMMAND>::=ASSIGN,<ASSIGN>
<COMMAND>::=SFILE,<LUN>,<INTEGER>
<COMMAND>::=SREC,<LUN>,<INTEGER>,F
<COMMAND>::=SREC,<LUN>,<INTEGER>,R
<COMMAND>::=WEOF,<LUN>
<COMMAND>::=REW,<LUN>
<COMMAND>::=PFILE,<LUN>,<KEY>,<NAME>
<COMMAND>::=FORM,<INTEGER>
<NAME>::=<LETTER>
<NAME>::=<NAME><LETTER>
<COMMENT>::=<NAME>
<INTEGER>::=<DIJIT>
<INTEGER>::=<INTEGER><DIJIT>
<ASSIGN>::=<LUN>=<NAME>
<ASSIGN>::=<LUN>=<LUN>
<KEY>::=<LETTER>
<LETTER>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<DIJIT>::=1|2|3|4|5|6|7|8|9|0
<LUN>::=<INTEGER>|$SI|$SO|$PI|$PO|$LO|$BI|$BO
```

APPENDIX D

RESULT OF TEACHING ERRORS MESSAGES FOR VORTEX

This Appendix illustrates teaching error messages for the subset of VORTEX given in Appendix C. The layout of the dialogue is the same as that in Appendix B. The last part of the session is an example of FIDO in operating mode.

```
+J
NO MORE MESSAGES
*"ALL JCP COMMANDS MUST START WITH A SLASH


+H
 1:"ALL JCP COMMANDS MUST START WITH A SLASH
*Y


+/
 1:"ALL JCP COMMANDS MUST START WITH A SLASH
*"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM


+/H
 2:"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
*Y


+/JOB F
 2:"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
*"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS


+/MEM 3
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*/ENDJOB F
*Y


+/ENDJOB F
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*"THE COMMAND ENDJOB HAS NO ARGUMENTS


+/SFILE 3
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*Y


+/ENDJOB 5
 4:"THE COMMAND ENDJOB HAS NO ARGUMENTS
*Y


+/JOB,
 2:"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
*"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>


 /JOB,FRED
ALL RIGHT


+/MEM,
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*"THE FORM OF THE MEM COMMAND IS : /MEM,<INTEGER>
```

```
+/FORM,
 6:"THE FORM OF THE MEM COMMAND IS : /MEM,<INTEGER>
*CHANGE 6
"BOTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS
*6


+/FORM,F
 6:"BOTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS
*Y


+/JOB,R2
 5:"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>
*"A NAME MUST CONTAIN LETTERS ONLY


+/C, TAKE 6
 7:"A NAME MUST CONTAIN LETTERS ONLY
*"A COMMENT CONTAINS LETTERS ONLY


+/PFILE
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>


+/PFILE 5
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*N
 4:"THE COMMAND ENDJOB HAS NO ARGUMENTS
*N
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*Y


+/PFILE,6,
 5:"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>
*N
 7:"A NAME MUST CONTAIN LETTERS ONLY
*N
 8:"A COMMENT CONTAINS LETTERS ONLY
*N
 2:"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
*N
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*Y


+/PFILE,1,E,C4
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*N
 7:"A NAME MUST CONTAIN LETTERS ONLY
*Y


+/PFILE
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*Y


+/PFILE,
 6:"BOTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS
*N
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*Y


+/PFILE,H
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*"<LUN> IS EITHER AN INTEGER OR A UNIT NAME


+/WEOF,$F
 5:"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>
*N
```

```
 7:"A NAME MUST CONTAIN LETTERS ONLY
*N
 8:"A COMMENT CONTAINS LETTERS ONLY
*N
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*N
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*N
 4:"THE COMMAND ENDJOB HAS NO ARGUMENTS
*N
 6:"BOTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS
*N
 2:"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
*N
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
*Y


+/PFILE,$T
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
*Y


+/SREC,6,6,
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*"THE FORM OF THE SREC COMMAND IS : /SREC,<LUN>,<INTEGER>,F OR R


+/SREC
11:"THE FORM OF THE SREC COMMAND IS : /SREC,<LUN>,<INTEGER>,F OR R
*Y


+/SREC,6,F
11:"THE FORM OF THE SREC COMMAND IS : /SREC,<LUN>,<INTEGER>,F OR R
*Y


+/SREC,$F
11:"THE FORM OF THE SREC COMMAND IS : /SREC,<LUN>,<INTEGER>,F OR R
*N
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
*Y


+
 1:"ALL JCP COMMANDS MUST START WITH A SLASH
*Y


+/JOB F
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*Y


 /JOB,FRED
ALL RIGHT
+/JOB, THIS IS A LONG JOB HEADER,
 7:"A NAME MUST CONTAIN LETTERS ONLY
*Y


+/
 2:"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
*LIST
 1:"ALL JCP COMMANDS MUST START WITH A SLASH
 2:"COMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
 4:"THE COMMAND ENDJOB HAS NO ARGUMENTS
 5:"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>
 6:"BOTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS
 7:"A NAME MUST CONTAIN LETTERS ONLY
 8:"A COMMENT CONTAINS LETTERS ONLY
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
11:"THE FORM OF THE SREC COMMAND IS : /SREC,<LUN>,<INTEGER>,F OR R
*Y
```

```
+/PFILE
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*Y


+/PFILE,T
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
*Y


+P
 1:"ALL JCP COMMANDS MUST START WITH A SLASH
*Y


+/PFILE,$C
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
*Y


+/PFILE,$SI,7
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*Y


+/PFILE,$SI,DF
 7:"A NAME MUST CONTAIN LETTERS ONLY
*"A KEY IS A SINGLE LETTER


+/PFILE,$SI,R7
 7:"A NAME MUST CONTAIN LETTERS ONLY
*N
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*N
 8:"A COMMENT CONTAINS LETTERS ONLY
*N
 5:"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>
*N
12:"A KEY IS A SINGLE LETTER
*Y


+/PFILE,$SI,7
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*N
12:"A KEY IS A SINGLE LETTER
*Y


+/PFILE,$SI,H, THIS 2
 7:"A NAME MUST CONTAIN LETTERS ONLY
*Y


+/PFILE,$SI,$
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*N
12:"A KEY IS A SINGLE LETTER
*Y


+/PFILE,3,4
 7:"A NAME MUST CONTAIN LETTERS ONLY
*N
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
*N
12:"A KEY IS A SINGLE LETTER
*Y


+/PFILE,3,$
12:"A KEY IS A SINGLE LETTER
*Y
```

```
+/PFILE,3,F,$
 7:"A NAME MUST CONTAIN LETTERS ONLY
*Y


 /REW,5
ALL RIGHT


+/SFILE,F
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>


+/SFILE,$F
13:"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>
*LIST
 1:"ALL JCP COMMANDS MUST START WITH A SLASH
 2:"COMMANDS ARE: JOB,ENDJOB,COMEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
 4:"THE COMMAND ENDJOB HAS NO ARGUMENTS
 5:"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>
 6:"BOTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS
 7:"A NAME MUST CONTAIN LETTERS ONLY
 8:"A COMMENT CONTAINS LETTERS ONLY
 9:"THE FORM OF THE PFILE COMMAND IS : /PFILE,<LUN>,<KEY>,<NAME>
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
11:"THE FORM OF THE SREC COMMAND IS : /SREC,<LUN>,<INTEGER>,F OR R
12:"A KEY IS A SINGLE LETTER
13:"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>
*10


+/SFILE,$I
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
*Y


+/SFILE
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*N
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME
*N
13:"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>
*Y


+/SFILE,
13:"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>
*Y


+/SFILE $
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
*Y


+/JOB,
 5:"THE FORM OF THE JOB COMMAND IS : /JOB,<NAME>
*Y


+
 1:"ALL JCP COMMANDS MUST START WITH A SLASH
*OPERATE


+/SFILE
13:"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>


+/JOB
 3:"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
```

```
+/JUB,
 5:"THE FURM UF THE JOB COMMAND IS : /JOB,<NAME>

+/JUB F
 3:"CUMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS

+/JUB/
 3:"CUMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS

+/JUB,G6
 7:"A NAME MUST CONTAIN LETTERS ONLY

+/
 2:"CUMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM

+S
 1:"ALL JCP CUMMANDS MUST START WITH A SLASH

+/H
 2:"CUMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FURM

+//
 2:"CUMMANDS ARE: JOB,ENDJOB,C,MEM,ASSIGN,SFILE,SREC,WEOF,REW,PFILE,FORM

+/SFILE
13:"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>

+/SFILE,
13:"SFILE COMMAND HAS THE FORM : /SFILE,<LUN>,<INTEGER>

+/SFILE F
 3:"CUMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS

+/SFIB
10:"<LUN> IS EITHER AN INTEGER OR A UNIT NAME

+/CA
 3:"CUMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS

+/JUE
 5:"THE FURM UF THE JOB COMMAND IS : /JOB,<NAME>

+/ENDJOB/
 4:"THE CUMMAND ENDJOB HAS NO ARGUMENTS

+/MEM N
 3:"CUMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS

+/MEM,N
 6:"BUTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS

+/FORM,Y
 6:"BUTH THE MEM AND FORM COMMANDS HAVE INTEGER ARGUMENTS
```

```
+/JOB,U2
 7!"A NAME MUST CONTAIN LETTERS ONLY

+/C,3
 8!"A COMMENT CONTAINS LETTERS ONLY

+/PFILE
 9!"THE FORM OF THE PFILE COMMAND IS ! /PFILE,<LUN>,<KEY>,<NAME>

+/PFILE,G
 9!"THE FORM OF THE PFILE COMMAND IS ! /PFILE,<LUN>,<KEY>,<NAME>
10!      "<LUN> IS

+/PFILE G
 3!"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS

+/PFILE,56
 9!"THE FORM OF THE PFILE COMMAND IS ! /PFILE,<LUN>,<KEY>,<NAME>

+/PFILE,U
 9!"THE FORM OF THE PFILE COMMAND IS ! /PFILE,<LUN>,<KEY>,<NAME>

+/PFILE,$U
10!"<LUN> IS EITHER AN INTEGER OR A UNIT NAME

+/SREC
11!"THE FORM OF THE SREC COMMAND IS ! /SREC,<LUN>,<INTEGER>,F OR R

+/SREC,U
10!"<LUN> IS EITHER AN INTEGER OR A UNIT NAME

+/SREC,$U
10!"<LUN> IS EITHER AN INTEGER OR A UNIT NAME

+/SREC,$SI,$
11!"THE FORM OF THE SREC COMMAND IS ! /SREC,<LUN>,<INTEGER>,F OR R

+/SREC,5,6,M
11!"THE FORM OF THE SREC COMMAND IS ! /SREC,<LUN>,<INTEGER>,F OR R

+/PFILE,4,JJ
12!"A KEY IS A SINGLE LETTER

+/PFILE,4,)
12!"A KEY IS A SINGLE LETTER

+/SFILE
13!"SFILE COMMAND HAS THE FORM ! /SFILE,<LUN>,<INTEGER>

+/SFILE L
 3!"COMMA NEEDED TO SEPERATE COMMAND AND ARGUMENTS
```

```
*/SFILE,3,F
101"<LUN> IS EITHER AN INTEGER OR A UNIT NAME

*
 1!"ALL JCP COMMANDS MUST START WITH A SLASH

*S
 1!"ALL JCP COMMANDS MUST START WITH A SLASH

*&
 1!"ALL JCP COMMANDS MUST START WITH A SLASH
```

APPENDIX E

TEACHING TALK TO ANSWER QUESTIONS ABOUT CANDE

This Appendix illustrates the use of FIDO in a question answering system. The program which performs feature extraction is called TALK, and is described in Chapter 6.4. The input to TALK is prompted by a "+". The input to FIDO is prompted by a "*". Apart from these and the second line of the "JOIN" and "CHANGE" commands, all other lines are output by FIDO.

```
+HOW DO I LOG ON
NO MORE MESSAGES
*"TO LOG ON TYPE "HELLO"


+HOW DO I LOG OFF
 1:"TO LOG ON TYPE "HELLO"
*"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"


+HOW DO I SIGN OFF
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*YES


+ARE THERE ANY ENQUIRY COMMANDS
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*"ENQUIRY COMMANDS START WITH A "?"


+HOW DO I ENTER ENQUIRY COMMANDS
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+WHAT ARE SOME USEFUL ENQUIRY COMMANDS
 3:"ENQUIRY COMMANDS START WITH A "?"
*"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ


+GIVE ME EXAMPLES OF ENQUIRY COMMANDS
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*Y


+GIVE ME ENQUIRY COMMAND EXAMPLES
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*YES


+IS THERE A CANDE COMMAND TO CLEAR THE LINE
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 1:"TO LOG ON TYPE "HELLO"
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
NO MORE MESSAGES
*"TO CLEAR THE LINE TYPE CONTROL B


+WHICH CANDE COMMAND TERMINATES A RUNNING PROGRAM
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N
 1:"TO LOG ON TYPE "HELLO"
*LIST
```

```
 1:"TO LOG ON TYPE "HELLO"
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
 3:"ENQUIRY COMMANDS START WITH A "?"
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
 5:"TO CLEAR THE LINE TYPE CONTROL B
*"TO TERMINATE A PROGRAM TYPE "?DS"


+WHAT DOES THE COMMAND HELLO DO
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
 1:"TO LOG ON TYPE "HELLO"
*Y


+CAN I LOG OFF WITH HELLO
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*Y


+HOW DO I INITIATE A CANDE SESSION
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 1:"TO LOG ON TYPE "HELLO"
*SC
*N
 5:"TO CLEAR THE LINE TYPE CONTROL B
*1


+IS THERE ANY TIME LIMIT ON A CANDE SESSION
 1:"TO LOG ON TYPE "HELLO"
*"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS


+HOW DO I TERMINATE A CANDE SESSION
 1:"TO LOG ON TYPE "HELLO"
*NO YOU ARE STUPID
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
*NO YOU GREAT APE
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*YES PHEW


+HOW DO I LIST THE CURRENT WORKFILE
 1:"TO LOG ON TYPE "HELLO"
*" TO LIST ANY FILE TYPE "LIST <FILENAME>"


+WHAT COMMAND SAVES THE WORKFILE
 6:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*"TO SAVE THE WORKFILE USE THE SAVE COMMAND


+CAN I KEEP MY WORKFILE FOR FUTURE REFERENCE
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N EHHHH?
 6:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*N
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*Y AH


+WHAT REFERENCE CAN I USE TO FAMILIARIZE WITH EDIT COMMANDS
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS


+IS THERE A LIST OF CANDE COMMANDS
 5:"TO CLEAR THE LINE TYPE CONTROL B
```

```
*N
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
*N
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
 1:"TO LOG ON TYPE "HELLO"
*10


+CAN U GIVE ME SOURCE LISTING CANDE COMMANDS
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*Y


+DO ALL CANDE COMMANDS START WITH "?"
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 1:"TO LOG ON TYPE "HELLO"
*N
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+"?"
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+ALL CANDE COMMANDS
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*Y
+WHAT COMMAND DO I USE TO TERMINATE A CANDE SESSION
 1:"TO LOG ON TYPE "HELLO"
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*Y


+WHICH CANDE COMMANDS START WITH  A "?"
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+HELP
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
*"WHAT FEATURE ARE YOU INTERESTED IN


+HOW DO I LOG-ON
 1:"TO LOG ON TYPE "HELLO"
*SC
*Y


+PLEASE GIVE OUTLINE OF CANDE SESSION
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
 1:"TO LOG ON TYPE "HELLO"
*L
 1:"TO LOG ON TYPE "HELLO"
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
 3:"ENQUIRY COMMANDS START WITH A "?"
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
 5:"TO CLEAR THE LINE TYPE CONTROL B
 6:"TO TERMINATE A PROGRAM TYPE "?DS"
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
```

```
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
11:"WHAT FEATURE ARE YOU INTERESTED IN
*"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILES


*WHAT DOES CANDE DO
 1:"TO LOG ON TYPE "HELLO"
*N
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*12


*ARE THERE ANY LIMITS ON DISK SPACE
 3:"ENQUIRY COMMANDS START WITH A "?"
*"CANDEPACK LIMITS ARE SET INDIVIDUALLY FOR EACH USERCODE


*CAN I INITIATE A BATCH RUN FROM CANDE
 1:"TO LOG ON TYPE "HELLO"
*"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND


*
 1:"TO LOG ON TYPE "HELLO"
*L
 1:"TO LOG ON TYPE "HELLO"
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
 3:"ENQUIRY COMMANDS START WITH A "?"
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
 5:"TO CLEAR THE LINE TYPE CONTROL B
 6:"TO TERMINATE A PROGRAM TYPE "?DS"
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
11:"WHAT FEATURE ARE YOU INTERESTED IN
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
13:"CANDEPACK LIMITS ARE SET INDIVIDUALLY FOR EACH USERCODE
14:"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND
*


*HOW DO I STOP A RUNNING PROGRAM
 6:"TO TERMINATE A PROGRAM TYPE "?DS"
*Y


*WHAT DOES THE SAVE COMMAND DO
 1:"TO LOG ON TYPE "HELLO"
*N
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*N
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*NAUGHTY ,NASTY
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*Y ITS THE WAY U HOLD YOUR FINGER


*WHAT DO I DO WHEN THERE IS NO RESPONSE
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*NOT RITE
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N HORSE
 1:"TO LOG ON TYPE "HELLO"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*5


*I AM GETTING NO RESPONSE
 5:"TO CLEAR THE LINE TYPE CONTROL B
*Y
```

```
+HOW DOES ONE GET STARTED ON THIS BEAST
13:"CANDEPACK LIMITS ARE SET INDIVIDUALLY FOR EACH USERCODE
*N
 1:"TO LOG ON TYPE "HELLO"
*SC
*Y


+HOW DO I LOG-OFF
 1:"TO LOG ON TYPE "HELLO"
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*2


+WHAT DO ENQUIRY COMMANDS LOOK LIKE
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+PLEASE GIVE EXAMPLES OF ENQUIRY COMMANDS
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*Y EXCELLENT


+THERE SEEMS TO BE NO RESPONSE TO ANY COMMANDS
 5:"TO CLEAR THE LINE TYPE CONTROL B
*Y


+HOW DO I ABORT MY PROGRAM
 6:"TO TERMINATE A PROGRAM TYPE "?DS"
*Y


+HOW LONG WILL CANDE LET MY PROGRAM RUN
 6:"TO TERMINATE A PROGRAM TYPE "?DS"
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
14:"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND
*N
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
 1:"TO LOG ON TYPE "HELLO"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*N
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
*Y


+CAN I LIST A FILE ON CANDEPACK
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*Y


+HOW CAN I PRESERVE MY FILE FOR A LATER SESSION
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*Y
```

```
+IS THERE A DOCUMENT LISTING ENQUIRY COMMANDS
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*Y


+PLEASE ASSIST ME
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*N
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*N
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
 1:"TO LOG ON TYPE "HELLO"
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 6:"TO TERMINATE A PROGRAM TYPE "?DS"
*N
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
*N
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*NN
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
11:"WHAT FEATURE ARE YOU INTERESTED IN
*Y


+WHAT THINGS CAN I DO ON CANDE
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
14:"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND
*N
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*N
 1:"TO LOG ON TYPE "HELLO"
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*Y


+HOW MUCH FILE SPACE AM I ALLLOWED
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
*N
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
13:"CANDEPACK LIMITS ARE SET INDIVIDUALLY FOR EACH USERCODE
*Y


+WHAT DOES THE START COMMAND INITIATE
 1:"TO LOG ON TYPE "HELLO"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
*N
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
 5:"TO CLEAR THE LINE TYPE CONTROL B
*N
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*N
14:"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND
*Y


+
 1:"TO LOG ON TYPE "HELLO"
*L
 1:"TO LOG ON TYPE "HELLO"
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
 3:"ENQUIRY COMMANDS START WITH A "?"
```

```
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
 5:"TO CLEAR THE LINE TYPE CONTROL B
 6:"TO TERMINATE A PROGRAM TYPE "?DS"
 7:"EXECUTION TIME IS LIMITED TO QUEUE 1 LIMITS
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
11:"WHAT FEATURE ARE YOU INTERESTED IN
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
13:"CANDEPACK LIMITS ARE SET INDIVIDUALLY FOR EACH USERCODE
14:"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND
*


+CAN I GET A PRINTOUT OF MY WORKFILE
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
 8:" TO LIST ANY FILE TYPE "LIST <FILENAME>"
*Y


+DO ALL CANDE COMMANDS START WITH A "?"
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+DO ALL CANDE COMMANDS START WITH A "?"
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+IS THERE A CANDE COMMAND TO DISPLAY THE STATUS OF MY PROGRAM
 5:"TO CLEAR THE LINE TYPE CONTROL B
*""?STA DISPLAYS PROGRAM STATUS


+DO ALL CANDE COMMANDS START WITH A "?"
 3:"ENQUIRY COMMANDS START WITH A "?"
*Y


+IS IT POSSIBLE TO TRIGGER A BATCH RUN FROM CANDE
14:"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND
*Y


+WHAT ARE SOME CANDE ENQUIRY COMMANDS
 4:"SOME USEFUL ENQUIRY COMMANDS ARE STA,CS,TI,SS,SQ
*Y


+PLEASE HELP ME
11:"WHAT FEATURE ARE YOU INTERESTED IN
*Y


+WHAT CAN I USE CANDE FOR
10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
14:"TO INITIATE A BATCH RUN FROM CANDE USE THE START COMMAND
*N GOT YOUR WIRES CROSSED
 9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*Y


+WHAT IS THE COMMAND TO INITIATE A CANDE SESSION
 1:"TO LOG ON TYPE "HELLO"
*Y


+HOW DO I START ON CANDE
12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*N
 2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 3:"ENQUIRY COMMANDS START WITH A "?"
```

```
*N
  1:"TO LOG ON TYPE "HELLO"
*Y


*HOW DO I FINISH ON CANDE
  2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*Y


*WHAT CAN I DO ON CANDE
 12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*Y


*CAN I STOP A RUNNING PROGRAM
  6:"TO TERMINATE A PROGRAM TYPE "?DS"
*Y


*CAN I EDIT PROGRAMS ON CANDE
  1:"TO LOG ON TYPE "HELLO"
*N
  2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*N
 10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*N
  9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*N
 12:"ON CANDE YOU CAN EDIT,COMPILE AND EXECUTE PROGRAMS AND MAINTAIN FILE
*Y


*I NEED TO KEEP MY WORKFILE FOR LATER
  9:"TO SAVE THE WORKFILE USE THE SAVE COMMAND
*Y


*WHERE CAN I GET A LIST OF CANDE COMMANDS
 10:"THE CANDE REFERENCE CARD CONTAINS ALL CANDE COMMANDS
*Y


*CAN I ENQUIRE ABOUT THE STATUS OF A PROGRAM
 15:""?STA DISPLAYS PROGRAM STATUS
*Y


*HOW DO I LOG OFF
  2:"TO LOG OFF TYPE EITHER "BYE" OR "HELLO"
*Y
```